

AD-A268 147



FINAL/30 SEP 89 TO 29 DEC 92

AN ADAPTIVE PLANNER FOR REAL-TIME  
UNCERTAIN ENVIRONMENTS (U)

Professor Paul Cohen

University of Massachusetts  
Computer Science Dept  
Amherst MA 01003DTIC  
ELECTE  
AUG 18 1993  
S C D7122/00/DARPA  
F49620-89-C-0113

AFOSR-TR- 93 0576

AFOSR/NM  
110 DUNCAN AVE, SUTE B115  
BOLLING AFB DC 20332-0001SPONSORING/MONITORING  
AGENCY REPORT NUMBER  
F49620-89-C-0113

## 11. SUPPLEMENTARY NOTES

## 12a. DISTRIBUTION AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

## 12b. DISTRIBUTION CODE

UL

## 13. ABSTRACT (Maximum 200 words)

The accomplishments under this contract were: (1) the researchers built an adaptive planning architecture for a complex, real-time task environment and a testbed for its principled analysis, (2) developed a model-based methodological approach and used it to analyze numerous aspects of the Phoenix agent architecture, (3) development of a procedure called failure recovery analysis (FRA), for analyzing execution traces of failure recovery to discover when and how the planner's actions may be causing failures, (4) extending the previous work with envelopes with the development of a simple one-parameter decision rule called a slack time envelope, (5) taking several steps toward a formalizing of the problem of plan execution monitoring, (6) building causal models of AI program behavior using path analysis and (7) expanding the scope of the methodological approach and authoring a textbook on empirical methods for Artificial Intelligence.

## 14. SUBJECT TERMS

93 8 12 02 7

15. NUMBER OF PAGES  
93

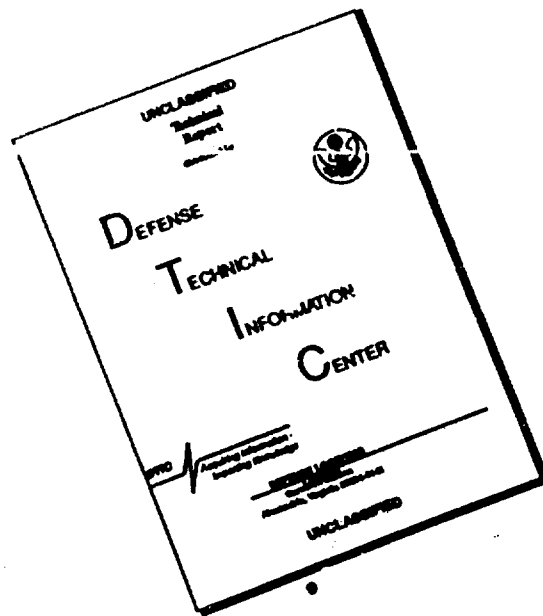
16. PRICE CODE

17. SECURITY CLASSIFICATION  
OF REPORT  
UNCLASSIFIED18. SECURITY CLASSIFICATION  
OF THIS PAGE  
UNCLASSIFIED19. SECURITY CLASSIFICATION  
OF ABSTRACT  
UNCLASSIFIED20. LIMITATION OF ABSTRACT  
SAR(SAME AS REPORT)

93-19132

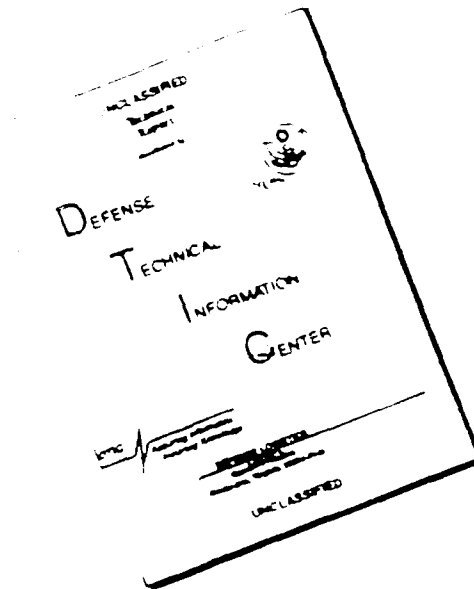


# DISCLAIMER NOTICE



**THIS DOCUMENT IS BEST  
QUALITY AVAILABLE. THE COPY  
FURNISHED TO DTIC CONTAINED  
A SIGNIFICANT NUMBER OF  
PAGES WHICH DO NOT  
REPRODUCE LEGIBLY.**

# DISCLAIMER NOTICE



THIS DOCUMENT IS BEST  
QUALITY AVAILABLE. THE COPY  
FURNISHED TO DTIC CONTAINED  
A SIGNIFICANT NUMBER OF  
PAGES WHICH DO NOT  
REPRODUCE LEGIBLY.

# An Adaptive Planner for Real-Time Uncertain Environments<sup>1</sup>

Paul Cohen, *Principal Investigator*  
David M. Hart, *Lab Manager*

Experimental Knowledge Systems Laboratory  
Computer Science Department  
University of Massachusetts  
Amherst, MA 01003  
413-545-3638

Final Technical Report  
Period 9/30/89 to 12/29/92

Accession For	
NTIS	CRA&I
DTIC	TAB
Unannounced	
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	2-3

DTIC TAB UNCLASSIFIED 3

Sponsored by Defense Advanced Research Projects Agency  
DARPA Order No. 7122, Program Code: 9E20  
Monitored by AFOSR Under Contract No. F49620-89-C-0113  
Contract Period: Sept. 30, 1989 to Dec. 29, 1992  
Amount of Contract Dollars: \$ 709,332

Dr. Abraham Waksman, *Program Manager*  
202-767-5025

<sup>1</sup> The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

## Abstract

Our goal under this three-year contract was to build an adaptive planner for a real-time, uncertain environment. The domain we chose was forest fire fighting, for which we built a simulator of forest fires and autonomous agents tasked to control them. In the first year of the contract we built a flexible agent architecture with a variety of adaptive mechanisms that makes Phoenix agents responsive to the changing demands of the task environment. We demonstrated that our multi-agent planning system built on this architecture can successfully fight simulated fires under a variety of circumstances.

In the second year of the contract we have focused on *why* our planning system works, and *whether it works well*. Our ongoing inquiry into the proper role of evaluation in AI system building led us to the development of a new approach to AI research – modeling and analyzing the relationship between the task environment and the agent design. Modeling AI architectures mathematically is a promising innovation that should provide the basis for much-needed evaluation and analysis. Building agents that "work" is not enough. To prove that we understand how they work, we must model them precisely enough to identify and correct the inevitable design inefficiencies. During the second year we applied this approach to our work in Phoenix, developing models of the fire-fighting task environment and Phoenix agent design that enabled us to form and test hypotheses about how our agents perform.

In this Final Technical Report we summarize results from the first two years, during which we 1) built an adaptive planning architecture for a complex, real-time task environment and a testbed for its principled analysis; and 2) developed a model-based methodological approach and used it to analyze numerous aspects of the Phoenix agent architecture. We then describe the five culminating accomplishments of our research in the third and final year:

1. development of a procedure we call *failure recovery analysis* (FRA), for analyzing execution traces of failure recovery to discover when and how the planner's actions may be causing failures;
2. extending our previous work with *envelopes* with the development of a simple one-parameter decision rule called a *slack time* envelope;
3. taking several steps toward a formalizing of the problem of plan execution monitoring;
4. building causal models of AI program behavior using *path analysis* ;
5. expanding the scope of our methodological approach and authoring a textbook on empirical methods for Artificial Intelligence.



**Table of Contents**

1. Numerical Productivity Measures .....	1
2. Executive Summary .....	3
2.1. Summary of Technical Results .....	3
2.2. Publications .....	15
2.3. Conferences, Workshops, Presentations .....	18
2.4. Awards, Promotions, Honors .....	21
2.5. Technology Transfer .....	23
2.6. Software Prototypes .....	26
2.7. References .....	27
3. Failure Recovery and <i>Failure Recovery Analysis</i> in Phoenix .....	29
4. An Empirical Method for Constructing <i>Slack time Envelopes</i> .....	63
5. Constructing an Envelope without a Model .....	69
6. Building Causal Models of Planner Behavior using <i>Path Analysis</i> .....	81
Appendix A: Textbook Prospectus (Empirical Methods for Artificial Intelligence) .....	91



# **1. Numerical Productivity Measures**

Refereed papers published: 16  
Refereed papers submitted: 3  
Invited papers published: 1  
Refereed workshop abstracts and symposia papers: 8  
Books or parts thereof published: 3  
Ph.D. dissertations: 1  
Unrefereed reports and articles: 10  
Invited presentations: 20  
Contributed presentations: 13  
Tutorials: 2  
Honors, including conference committees: 8  
Graduate students supported at least 25% time: 8



## **2. Executive Summary**

### **2.1. Summary of Technical Results**

#### **2.1.1. Accomplishments in the First Two Contract Years**

Our original goal under this contract was to build a real-time, adaptive planner, based on an agent architecture capable of integrating multiple planning methods. The problem domain we chose was forest fire fighting. We built a simulator of forest fires and autonomous agents tasked to control them. This system, which we called Phoenix, consists of an instrumented discrete event simulation, an architectural shell for autonomous agents that integrates multiple planning methods, and an organization of planning agents capable of improving their fire-fighting performance by adapting to the simulated environment.

Our original approach to this large research endeavor was, 1) to build a realistic simulated world, 2) build simulated autonomous agents to solve problems in the world, and 3) conduct experiments designed to demonstrate that our solution "worked". Thus, in the first year of this contract we designed a flexible agent architecture with a variety of mechanisms supporting the following: delayed commitment of resources in the face of a dynamic environment; real-time control of cognitive processes; sophisticated monitoring of change and progress; the ability to react very quickly (reflexively) to sudden changes in the world; and learning (Cohen et al. 1989). However, during this first year our ongoing inquiry into the proper role of evaluation in AI system building (Cohen 1991b) led us to the development of a new approach to AI research. We still believe in the importance of steps 1 and 2 above, but would substitute the following steps for the third: 3) analyze the task environment and the design of the agents by modeling them mathematically; 4) use the models to predict the performance of proposed designs; 5) verify predicted performance and identify design optimizations empirically; 6) implement the optimized designs and test them.

Modeling AI architectures mathematically is a promising innovation that should provide the basis for much-needed evaluation and analysis (Cohen 1991b). Building agents that "work" is not enough. To prove that we understand how they work, we must model them precisely enough to identify and correct the inevitable design inefficiencies. In the second year of the contract we applied this approach to our work in Phoenix, developing models of the Phoenix task environment and agent design that enable us to form and test hypotheses about how our agents perform. A sampling of these modeling efforts includes:

- Refining our view of modeling and the critical role we feel it plays in putting AI on a firm scientific footing.

- Developing cost models for recovery from plan failures (Howe & Cohen 1991), along with an accompanying analysis of several problems that arose from this cost model.
- Developing models of wind dynamics and their effect on fire spread (Hansen 1990a, Hansen 1990b). Models such as these capture the kinds of environmental constraints imposed on agents operating in this domain.
- Developing several related models of optimal fire-fighting strategies in Phoenix. These models generated hypotheses that we subsequently tested in large empirical trials (Cohen, Hart & Devadoss 1990).
- Developing an analysis of the utility of *envelopes* in Phoenix by examining the time interval between monitoring events (Anderson & Hart 1990).
- Analyzing an abstracted model of the cognitive scheduling problem for the fireboss agent in Phoenix (Anderson, Hart & Cohen 1991).

These are described in detail in the 1992 Annual Technical Report.

### 2.1.2. Accomplishments in the Final Year

Our work in the third (final) year of this contract produced five important accomplishments:

1. developing a procedure we call *failure recovery analysis* (FRA), for analyzing execution traces of failure recovery to discover when and how the planner's actions may be causing failures;
2. extending our previous work with *envelopes* with the development of a simple one-parameter decision rule called a *slack time* envelope;
3. taking several steps toward a formalizing of the problem of plan execution monitoring;
4. building causal models of AI program behavior using *path analysis*;
5. expanding the scope of our methodological approach and authoring a textbook on empirical methods for Artificial Intelligence.

Each of these accomplishments is discussed in turn in Sections 2.1.3 – 2.1.7.

### 2.1.3. Failure Recovery Analysis

The third year of the contract saw the completion of Adele Howe's thesis on failure recovery in Phoenix, in which she developed a new approach to debugging AI planning systems (which also can be extended for debugging other large AI systems as well). The development of this approach can be traced through many of the papers sponsored by this contract, including Howe & Cohen (1990), Howe & Cohen (1991), Howe (1992), and Howe (1993). Several articles have been recently submitted on this subject, including one to the *Artificial Intelligence Journal* about evaluating AI planner behavior, and to *IEEE Transactions on Knowledge and Data Engineering* about generalizing this approach to debugging failures in large software systems. This approach is briefly outlined in this section, and more fully explained in Section 3.

Plans fail for perfectly good reasons: the environment changes unpredictably, sensors return flaky data, and effectors do not work as expected. During planner development, plans fail for not so good reasons: the effects of actions are not adequately specified,

apparently unrelated actions interact, and the domain model is incomplete and incorrect. Planners should not *cause* their own failures, but figuring out what went wrong and preventing it later is not easy. Failures tell us what went wrong, but not why. The failure repair alleviates the immediate problem, but does not tell us how to fix the cause or even whether the repair itself might not cause failures later. We have developed a procedure, called *failure recovery analysis* (FRA), for analyzing execution traces of failure recovery to discover when and how the planner's actions may be causing failures (Howe 1993).

Most approaches to debugging planners are knowledge intensive, assuming that the planner or debugger has a strong model of the domain. The approach we have developed, FRA, requires little knowledge to identify contributors to failures and only a weak model to explain how the planner might have caused failures. Complementary to the more knowledge intensive approaches, this approach is most appropriate when a rich domain model is not available or when the existing model might be incorrect or buggy, as when the system is under development.

The consequence of relying on a weak model is that while FRA can detect possible causes of the failure, it cannot identify *the* cause precisely enough to implement a repair. Debugging a planner requires judgment about what would be the best modification and whether the failure is worth avoiding at all. In repairing one failure, others might be introduced. In FRA, the designer decides how best to repair the failures.

Failure recovery analysis involves four steps:

1. execution traces are searched for statistically significant dependencies between recovery efforts and subsequent failures;
2. dependencies are mapped to structures in the planner's knowledge base known to be susceptible to failure;
- 3 the interactions and vulnerable plan structures are used to generate explanations of why the failures occur;
4. the explanations serve to separate occasional, acceptable failures from chronic, unacceptable failures, and recommend redesigns of the planner and recovery component.

These steps are described in detail and examples of their use in Phoenix given in Section 3.

Analyses of failure recovery can contribute in several ways to our understanding of planner performance. FRA can identify contributors to failure and assist in the debugging and evaluation of planners with incomplete or incorrect domain models. Additionally, the dependencies provide a measure of similarity between test situations. The more the environment and agent change, the more one expects observed effects to change; thus, dependencies can be a kind of similarity measure across planners and environments.

The lesson from this analysis is that while design changes rarely have isolated effects, designers do not have to give up hope of analyzing the effects. They can track the effects: They make minor changes and havoc ensues, but they have a way to assess the havoc. Phoenix is an example of a system that can interleave plans in arbitrary ways, as dictated by situation. Debugging its failures by "watching the system" or by predicting all possible execution traces is simply not feasible, but running Phoenix many times and analyzing the data is feasible. Failure recovery analysis isolates indirect effects of design changes and proposes explanations and modifications based on a weak model of the planner and its environment; its primary contribution is in helping us understand how planning decisions and actions interact, and assisting in debugging planners under development.

#### 2.1.4. Slack time Envelopes

During the third contract year we extended our work on the real-time monitoring and control structure we call *envelopes*. We showed previously that envelopes could be applied in Phoenix to compare the *actual* progress of plans to the *expected* progress on which the plans were based (Hart et al., 1990). Such a comparison can be used to predict plan failure in advance, thus allowing the planning system a head start in responding to failure. More recently we have shown that good performance can be achieved by hand constructed slack time envelopes (Cohen, St. Amant & Hart 1992), and we presented a probabilistic model of progress, from which we derived a method for automatically constructing slack time envelopes that balances the benefits of *early warning* that a plan is failing against the costs of *false positives* (erroneous predictions that a plan is failing caused by uncertainty in our predictions or serendipitous events).

Underlying the judgment that a plan will not succeed is a fundamental tradeoff between the cost of an incorrect decision and the cost of evidence that might improve the decision. For concreteness, let's say a plan succeeds if a vehicle arrives at its destination by a deadline, and fails otherwise. At any point in a plan we can correctly or incorrectly predict that the plan will succeed or fail. If we predict early in the plan that it will fail, and it eventually fails, then we have a hit, but if the plan eventually succeeds we have a false positive. False positives might be expensive if they lead to replanning. In general, the false positive rate decreases over time (e.g., very few predictions made immediately before the deadline will be false positives) but the reduction in false positives must be balanced against the cost of waiting to detect failures. Ideally, we want to accurately predict failures as early as possible; in practice, we can have accuracy or early warnings but not both.

The false positive rate for a decision rule that at time  $t$  predicts failure will generally decrease as  $t$  increases. In work detailed in Section 4, we analyze this tradeoff in several ways. First, we describe a very simple decision rule, called a *slack time envelope*, that we have used for years in the Phoenix planner. Then, using empirical data from Phoenix, we evaluate the false positive rate for envelopes and show that envelopes can maintain good performance throughout a plan. An infinite number of slack time envelopes can be constructed for any plan, and the first analysis in the

paper depends on "good" envelopes constructed by hand. To be generally useful, envelopes should be constructed automatically. This requires a formal model of the tradeoff between when a failure is predicted (earlier is better) and the false positive rate of the prediction, shown next in Section 4. Finally we show how the conditional probability of a plan failure given the state of the plan can be used to construct "warning" envelopes.

Although we rely heavily on slack time envelopes in the Phoenix planner, we have always constructed them by heuristic criteria, and we did not know how to evaluate their performance. In this work we showed that good performance can be achieved by hand-constructed slack time envelopes, and we presented a probabilistic model of progress, from which we derived a method for automatically constructing slack time envelopes that balances the benefits of early warnings against the costs of false positives. Our contribution has been to cast the problem in probabilistic terms and to develop a framework for evaluation. We are presently extending our work to other models of progress and different, more complex domains.

### 2.1.5. Timing is Everything: A Theoretical Look at Plan Execution Monitoring

The value of actions depends on their timing. Actions that interact with processes are more or less effective depending on when they occur. We are familiar with the idea of a window of opportunity, but it isn't always clear how to recognize such a window before it closes. Sometimes it is advantageous to monitor processes to detect windows. In (Cohen 1991a) we describe several timing problems; some require monitoring and some don't. We consider two monitoring strategies, a periodic strategy for monitoring for fires in Phoenix, and a monitoring strategy for predicting when a task will finish. We have proved the optimality of the former, and "mature" humans engage in the latter, although it isn't necessarily optimal. We also describe one case in which the cost functions for two processes are combined. Cohen (1991a) ends by suggesting that the large and somewhat bewildering range of timing problems might be described by relatively few features.

These features can be composed to form what we think is a preliminary *taxonomy of monitoring problems*. We have begun to establish this taxonomy, and have set about tackling some of the constituent monitoring problems. Some of these efforts are described in this section. First we report on a survey of the literature on plan execution monitoring. Next we discuss an optimal (though expensive) monitoring strategy for predicting whether a task will meet a deadline (the envelope problem). This strategy works only if we have a model of the process being monitored. In the final part of this section we discuss a method for *learning* an optimal monitoring strategy for tasks with deadlines when no model of the task is available.

**Monitoring Plan Execution: A Short Survey.** In Hansen & Cohen (1992c) we survey the progress that has been made on the problem of monitoring plan execution in the twenty years since the first, simple scheme was developed (Fikes 1971; Fikes, Hart & Nilsson, 1972). Although the research team that built the Shakey robot gave as much attention to the problem of execution monitoring as they did to the problem of plan

generation, for more than a decade afterward the research community focused almost exclusively on the problem of plan generation. The problem of execution monitoring was regarded, at best, as a side-issue. However the last few years have seen a revived interest in plan execution systems, an interest spurred by the desire to build agents that can operate effectively in complex, changing environments. With this in mind, it seems worthwhile to collect together in one place pointers to the scattered work that has been done on this subject, to provide some structure to it, and to analyze the issues it raises. While many good surveys of plan generation have been written, until now no comparable survey has been made of the work that has been done on execution monitoring.<sup>2</sup>

As a broad characterization, execution monitoring is a way of dealing with uncertainty about the effect of executing a plan. In general, there are two reasons for monitoring plan execution. The more basic one, which might be called "monitoring for failures", simply consists of checking that a plan works, that actions have their intended effect. The second might be called "monitoring for opportunities". It involves checking for things that need to be done, for events that need to be responded to. It can also involve checking for shortcuts or "optimizations" to a plan that may become available as the plan executes. In this case, the question is not whether a plan works but whether it could be made better.

This survey addresses the following broad questions:

- *What to Monitor.* We begin by considering the question of what to monitor. An answer is to monitor what is relevant, and to determine what is relevant by using the dependency structure of the plan.
- *When to Monitor.* There are two ways in which the question of when and how often to monitor has been addressed. One is by analyzing the uncertainty introduced into a plan by an agent's own actions, and triggering a monitoring action whenever the cumulative uncertainty exceeds some threshold. The other is by modeling the rate of change of the process in the environment being monitored, and setting the monitoring frequency to reflect that rate of change.
- *Monitoring and Sensing.* There is a close relationship between monitoring and sensing, so much so that it can seem natural to identify the two, to say they are one and the same. However in a number of schemes for execution monitoring, monitoring and sensing are distinguished.
- *Architectures for Monitoring.* After investigating the question of what conditions to monitor, as well as when and how often to monitor them, we look at the relationship between monitoring and sensing. The latter question brings us to the point of considering how schemes for monitoring affect, and are affected by, the design of an agent architecture, a subject we could refer to as "how to monitor".

**Monitoring to Predict Whether a Task Will Meet a Deadline.** Let us consider the problem of checking a decision rule that predicts whether a task will finish by a

---

<sup>2</sup> This survey, to appear in *AI Magazine*, is part of Eric Hansen's Master's project. Ongoing work on monitoring is being supported under an Augmentation Award for Science and Engineering Research Training.

deadline. One instance of this is envelopes, and is implemented as part of our Phoenix planner (Hart, Anderson & Cohen, 1990). The idea of using a decision rule such as an envelope to anticipate failure to meet a deadline soon enough ahead of time to initiate recovery has wide application, especially for real-time computing. AI systems that do approximate processing under time pressure monitor progress so that they can adjust their processing strategy to make sure they generate at least an approximate solution by a deadline (Lesser, Pavlin & Durfee, 1988). Similarly, dynamic schedulers for real-time operating systems monitor task execution so that they can anticipate failure to meet a task deadline as soon as possible and take appropriate action (Haben & Shin, 1990).

Given a decision rule (such as an envelope) that predicts whether or not a deadline will be met, it remains to be decided *how often this rule should be tested* (in other words, how often should the envelope be monitored). If monitoring has no cost, it can be tested continuously. But if it has a cost, there must be a scheduling policy for it. When we built the Phoenix planner we assumed a periodic strategy for monitoring. This was purely ad hoc; we monitored the envelope to check performance every fifteen minutes of simulated time, without regard for the cost of making each check. This is also true in Haben & Shin's dynamic scheduling system, which assumes there is no cost of monitoring.

To solve what we will call the "envelope monitoring problem" using stochastic dynamic programming, we first express it in formal mathematical terms. Its state is represented by a vector of two variables: 1) the time remaining before the deadline, and 2) the distance remaining to reach the goal condition. There is a single decision variable,  $m$ ; the decision is to either stop and abandon the task or to continue executing it for  $m$  additional units of time before monitoring its progress again. The complete solution is given in Hanson (1993). The obvious qualitative observation to make is that the frequency of monitoring increases with closeness to the envelope boundary. This supports the intuition that the more likely one is to cross the threshold of a decision rule, the more often one should check (or "monitor") the rule.

We are currently working to reconcile this optimal but costly strategy (time complexity  $O(n^3)$  and space complexity  $O(n^2)$ ) with the one-parameter slack-time envelope boundary estimator discussed in Section 2.1.4. Our intuition is that the one-parameter rule can serve as a cheap approximation of the optimal strategy at times when there is insufficient time to compute it.

**Learning a Decision Rule for Monitoring Tasks with Deadlines.** In the preceding section (and in Hanson 1993) we showed that, given a model of the state transition probabilities and payoffs, an optimal monitoring policy can be determined using stochastic dynamic programming. In Hanson and Cohen (1992a) we extend this work, showing that even without a model, an optimal monitoring policy can be *learned*.

A real-time scheduler or planner is responsible for managing tasks with deadlines. When the time required to execute a task is uncertain, it may be useful to monitor the task to predict whether it will meet its deadline; this provides an opportunity to make adjustments or else to abandon a task that can't succeed. Hanson (1993) treats monitoring a task with a deadline as a sequential decision problem. Given an explicit model of task execution time, execution cost, and payoff for meeting a deadline, an optimal decision rule for monitoring the task can be constructed using stochastic dynamic programming. If a model is not available, the same rule can be learned using *temporal difference methods* (Barto, Sutton & Watkins, 1990). These results are significant because of the importance of this decision rule in real-time computing.

It makes sense to construct a decision rule such as the one described in Hanson (1993) for tasks that are repeated many times or for a class of tasks with the same behavior. This allows the rule to be learned, if TD methods are relied on; or for statistics to be gathered to characterize a probability and cost model, if dynamic programming is relied on. However if a model is known beforehand, or can be estimated, a decision rule can also be constructed for a task that executes only once.

The time complexity of the dynamic programming algorithm is  $O(n^2)$ , where  $n$  is the number of time steps from the start of the task to its deadline; however the decision rule may be compiled once and reused for subsequent tasks. The time complexity of TD learning,  $O(n)$ , is mitigated by the possibility of turning learning off and on. The space overhead of representing an evaluation function by a table is avoidable by using a more compact function representation, such as a connectionist network.

Besides the fact that this approach is not computationally intensive, it has other advantages. It is conceptually simple. The decision rule it constructs is optimal, or converges to the optimal in the case of TD learning. It works no matter what probability model characterizes the execution time of a task and no matter what cost model applies, and so is extremely general. Finally, it works even when no model of the state transition probabilities and costs is available, although a model can be taken advantage of.

These results can be extended in two obvious ways. The first is to factor in a cost for monitoring. Our analysis thus far has assumed that monitoring has no cost, or its cost is negligible. This allows monitoring to be nearly continuous, in effect, for a task to be monitored each time step. Others who have developed similar decision rules have also assumed that the cost of monitoring is negligible. However in some cases the cost of monitoring may be significant, thus we show in another paper how this cost can be factored in (Hansen 1993). Once again we use dynamic programming and TD methods to develop optimal monitoring strategies.

The second way in which this work can be extended is to make the decision rule more complicated. Here we analyzed a simple example in which the only alternative to continuing a task is to abandon it. But recovery options may be available as well. A dynamic scheduler for a real-time operating system is unlikely to have recovery

options available, but an AI planner or problem-solver is almost certain to have them (Lesser, Pavlin & Durfee, 1988; Howe, 1992). The way to handle the more complicated decision problem this poses is to regard each recovery option as a separate task characterized by its own probability model and cost model; so at any point the expected value of the option can be computed. Then, instead of choosing between two options, either continuing a task or abandoning it, the choice includes the recovery options as well. The rule is simply to choose the option with the highest expected value.

The work described in this and the preceding subsection constitutes the second half of Eric Hansen's Master's project. Section 5 presents a more detailed overview of this work.

#### **2.1.6. Causal Modeling using Path Analysis**

During the third contract year we decided to baseline the real-time performance of the Phoenix Fireboss to help us design real-time scheduling algorithms for its cognitive activities. We undertook the experiment described in Section 6 (Hart & Cohen 1992) to measure how changes in the Fireboss's thinking speed affected its real-time performance. To analyze the results we used a statistical modeling technique called path analysis that we feel holds great promise as a method for building causal models from empirical observations of AI planner behavior.

It is difficult to predict or even explain the behavior of any but the simplest AI programs. A program will solve one problem readily, but make a complete hash of an apparently similar problem. For example, our Phoenix planner, which fights simulated forest fires, will contain one fire in a matter of hours but fail to contain another under very similar conditions. We therefore hesitate to claim that the Phoenix planner "works." The claim would not be very informative, anyway — we would much rather be able to predict and explain Phoenix's behavior in a wide range of conditions (Cohen 1991b). In Section 6 we describe an experiment with Phoenix in which we uncover factors that affect the planner's behavior and test predictions about the planner's robustness against variations in some factors (Hart & Cohen 1992). We also introduce a technique—path analysis—for constructing and testing causal explanations of the planner's behavior. Our results are specific to the Phoenix planner and will not necessarily generalize to other planners or environments, but our techniques are general and should enable others to derive comparable results for themselves.

We designed an experiment with two purposes. A confirmatory purpose was to test predictions that the planner's performance is sensitive to some environmental conditions but not others. In particular, we expected performance to degrade when we change a fundamental relationship between the planner and its environment—the amount of time the planner is allowed to think relative to the rate at which the environment changes—and not be sensitive to common dynamics in the environment such as weather, and particularly, wind speed. We tested two specific predictions: 1) that performance would not degrade or would degrade gracefully as wind speed increased; and 2) that the planner would not be robust to changes in the Fireboss's

thinking speed due to a bottleneck problem described below. An exploratory purpose of the experiment was to identify the factors in the Fireboss architecture and Phoenix environment that most affected the planner's behavior, leading to a causal model of the time required to put out a fire.

In order to illustrate the usefulness of path analysis for modeling causal relationships, it is necessary to delve a little bit into the workings of the Phoenix planner. The Fireboss must select plans, instantiate them, dispatch agents and monitor their progress, and respond to plan failures as the fire burns. The rate at which the Fireboss thinks is determined by a parameter called the Real Time Knob. By adjusting the Real Time Knob we allow more or less simulation time to elapse per unit CPU time, effectively adjusting the speed at which the Fireboss thinks relative to the rate at which the environment changes.

The Fireboss services bulldozer requests for assignments, providing each bulldozer with a task directive for each new fireline segment it builds. The Fireboss can become a bottleneck when the arrival rate of bulldozer task requests is high or when its thinking speed is slowed by adjusting the Real Time Knob. This bottleneck sometimes causes the overall digging rate to fall below that required to complete the fireline polygon before the fire reaches it, which causes replanning. In the worst case, a Fireboss bottleneck can cause a thrashing effect in which plan failures occur repeatedly because the Fireboss can't assign bulldozers during replanning fast enough to keep the overall digging rate at effective levels. We designed our experiment to explore the effects of this bottleneck on system performance and to confirm our prediction that performance would vary in proportion to the manipulation of thinking speed. Because the current design of the Fireboss is not sensitive to changes in thinking speed, we expect it to take longer to fight fires and to fail more often to contain them as thinking speed slows.

In contrast, we expect Phoenix to be able to fight fires at different wind speeds. It might take longer and sacrifice more area burned at high wind speeds, but we expect this effect to be proportional as wind speed increases and we expect Phoenix to succeed equally often at a range of wind speeds, since it was designed to do so.

In Section 6 we show that performance did indeed degrade as we systematically slowed Fireboss thinking speed. Interestingly, this degradation was not linear (with respect to the time required to contain the fire). We tried using multiple regression to model the factors that determine this nonlinear relationship, but found that while we could derive a predictive model, such a regression model doesn't allow us to explain the inter-related causal influences among the factors. We were able to apply path analysis (Li 1975; Asher 1983) to build a model that is both predictive and explanatory, and which tells us (among other things) how Phoenix performance will be affected by changes in the amount of thinking time available to the Fireboss.

Path analysis is a generalization of multiple linear regression that builds models with causal interpretations. It is an *exploratory* or *discovery* procedure for finding causal

structure in correlational data. In the months since this contract has terminated we have continued this work, applying path analysis to the problem of building models of AI programs, which are generally complex and poorly understood. Path analysis has a huge search space, however. If one measures  $N$  parameters of a system, then one can build  $O(2^{N^2})$  causal models relating these parameters. For this reason, we are currently developing an algorithm that heuristically searches the space of causal models.

### 21.7. Continuing Methodological Development

The design of AI systems is typically justified informally. For example, one might say, "The planner is designed to be reactive because the environment changes rapidly in unexpected ways." We believe this style of justification is too informal to support (a) demonstrations of the necessity of a design, (b) evaluation of the design, (c) generalization of the design to other tasks and environments, (d) communication of the design to other researchers, and (e) comparisons between designs. Through our research program we seek to demonstrate that achieving these goals is a natural consequence of basing designs on formal models of the interactions between agents and their environments. The methodology we have developed for this purpose we call *modeling, analysis and design* (MAD) (Cohen 1991b).

While the development of this methodology was funded under other contracts, we have consistently applied it to our work in Phoenix (see previous Annual Technical Reports). This provides us with a rich source of examples, from models of the task environment (Hansen 1990a, Hansen 1990b, Silvey 1990) to models of the Phoenix agent architecture (Cohen 1990, Anderson et al. 1991, Hart & Cohen 1992) to the design of experiments to evaluate planning system behavior (Howe & Cohen 1991). These examples have in turn been incorporated into presentations of our methodological approach in numerous forums, from conferences (see Conferences, Workshops and Presentations) to magazine articles (Cohen 1991b), and finally to a textbook and graduate course curriculum on AI methodology (Cohen forthcoming). Some examples of these activities include:

- *Presentations*. Paul Cohen was invited to deliver keynote addresses on methodological issues at a conference and a AAAI Spring Symposium (see Invited Presentations). He also participated in the Workshop on Research in Experimental Computer Science, the goal of which was to identify issues and problems arising in experimental work in the entire field of Computer Science. Sponsored by ONR, DARPA, and NSF, this workshop was held in Palo Alto, CA, October 16-18, 1991.
- *Workshop on AI Methodology*. Held in June of 1991, this workshop, sponsored jointly by DARPA and NSF, brought together leading AI researchers to discuss growing methodological concerns and develop a consensual strategy for addressing them.
- *Agentology Curriculum*. During the summer of 1991 we conducted a summer school designed to develop the skills in our graduate students needed to conduct MAD research, and believe that this effort has laid the groundwork for a curriculum in *agentology* -- the principled design of autonomous agents for complex environments. From that summer school we have developed a

research methods course for AI graduate students and are working on an accompanying textbook on Experimental Methods for AI Research.

- *AAAI Tutorial on Experimental Methods for AI Research*. This tutorial was offered jointly with Prof. Bruce Porter (Univ. of Texas, Austin) at AAAI-92, and will be offered again at AAAI-93.

**A Textbook for Empirical Methods in Artificial Intelligence.** The activities listed above are culminating now in a textbook being prepared for use in graduate AI methods courses. Entitled "Empirical Methods for Artificial Intelligence," this textbook is a primer for the empirical evaluation of the new generation of agents being designed by AI researchers. A prospectus for the textbook appears in Appendix A.

## 2.2. Publications

### 2.2.1. Refereed Papers Published

- Anderson, S.D., Hart, D.M. & Cohen, P.R. Two ways to act. *AAAI Spring Symposium on Integrated Intelligent Architectures*. Published in the *SIGART Bulletin*, 2(4):20-24. 1991.
- Cohen, P.R. & Hart, D.M. Path analysis models of an autonomous agent in a complex environment. To appear in *Proceedings of the Fourth International Workshop on AI and Statistics*. 1993.
- Cohen, P.R., St. Amant, R. & Hart, D.M. Early warning of plan failure, false positives and envelopes: Experiments and a model. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates, Inc. 1992. Pp. 773-778.
- Cohen, P.R. A Survey of the Eighth National Conference on Artificial Intelligence: Pulling together or pulling apart? *AI Magazine*, 12(1), 16-41.
- Cohen, P.R. Designing and analyzing strategies for Phoenix from models. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, Katia Sycara (Ed.). Morgan-Kaufman, 1990. Pp. 9-21.
- Cohen, P.R., Greenberg, M.L., Hart, D.M., & Howe, A.E. Real-Time problem solving in the phoenix environment. *Proceedings of the Workshop on Real-time Artificial Intelligence Problems at the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989.
- Cohen, P.R., Greenberg, M.L., Hart, D.M., & Howe, A.E. Trial by fire: Understanding the design requirements for agents in complex environments. Reprinted in *Nikkei Artificial Intelligence*, 102-119, Nikkei Business Publications, Inc., 1990. (Originally published in *AI Magazine*, 32-48, Fall 1989.)
- Hart, D.M. & Cohen, P.R. Predicting and explaining success and task duration in the Phoenix planner. *Proceedings of the First International Conference on AI Planning Systems*. Morgan Kaufmann. 1992. Pp. 106-115.
- Hart, D.M., Anderson, S.D., & Cohen, P.R. Envelopes as a vehicle for improving the efficiency of plan execution. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, Katia Sycara (Ed.). Morgan-Kaufman, 1990. Pp. 71-76.
- Howe, A.E. & Cohen, P.R. Detecting and explaining dependencies in execution traces. To appear in *Proceedings of the Fourth International Workshop on AI and Statistics*. 1993.
- Howe, A.E. Isolating dependencies on failure by analyzing execution traces. *Proceedings of the First International Conference on AI Planning Systems*. Morgan Kaufmann. 1992. Pp. 277-278.
- Howe, A.E. Analyzing failure recovery to improve planner design. *Proceedings of the Tenth National Conference on Artificial Intelligence*. AAAI Press/MIT Press. 1992. Pp. 387-392.
- Howe, A.E. & Cohen, P.R. Failure recovery: A model and experiments. *Proceedings of the Ninth National Conference on Artificial Intelligence*. Pasadena, CA. July 1991. Pp. 801-808.
- Howe, A.E., Hart, D.M. & Cohen, P.R. Addressing real-time constraints in the design of autonomous agents. *The Journal of Real-Time Systems*, 1(1/2):81-97. 1990.

Howe, A.E. & Cohen, P.R. Responding to environmental change. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, Katia Sycara (Ed.). Morgan-Kaufman, 1990. Pp. 85-92.

Powell, Gerald M. and Cohen, Paul R. Operational planning and monitoring with envelopes. *Proceedings of the Fifth Annual AI Systems in Government Conference*. 1990.

## 2.2.2. Refereed Papers Submitted

Hanks, S., Pollack, M.E. & Cohen, P.R. Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. To appear in *AI Magazine*.

Howe, A.E. Improving the reliability of AI planning systems by analyzing their failure recovery. Submitted to *IEEE Transactions on Knowledge and Data Engineering*.

Howe, A.E. & Cohen, P.R. Understanding planner behavior. Submitted to the *Artificial Intelligence Journal Special Issue on Planning and Scheduling* (D. McDermott & J. Hendler, eds.).

## 2.2.3. Invited Papers Published

Cohen, P.R. Methodological problems, a model-based design and analysis methodology, and an example. *Proceedings of the International Symposium on Methodologies for Intelligent Systems*. Pp. 33-50. Knoxville, TN, Oct. 25-27, 1990.

## 2.2.4. Refereed Workshop Abstracts and Symposia Papers

Cohen, P.R., Anderson, S.D., Hart, D.M. Scheduling agent actions in real-time. Abstract for *The Interdisciplinary Workshop on the Design Principles for Real-Time Knowledge Based Control Systems* at the Eighth National Conference on Artificial Intelligence. Boston, MA, 1990.

Cohen, P.R. & Howe, A.E. Benchmarks are not enough; Evaluation metrics depend on the hypothesis. *Collected Notes from the Benchmarks and Metrics Workshop*. Technical Report FIA-91-06, NASA Ames Research Center. Pp. 18-19. 1990.

Hart, David M. and Cohen, Paul R. Phoenix: A testbed for shared planning research. *Collected Notes from the Benchmarks and Metrics Workshop*. Technical Report FIA-91-06, NASA Ames Research Center. Pp. 20-27. 1990.

Howe, A.E. Evaluating planning through simulation: An example using Phoenix. *Working Notes of AAAI Spring Symposium on Foundations of Classical Planning*. Palo Alto, CA. March 1993.

Howe, A.E. Failure Recovery Analysis as a tool for plan debugging. In *Working Notes of the AAAI Spring Symposium on Computational Considerations in Supporting Incremental Modification*. Palo Alto, CA. March 26-27, 1992.

Howe, A.E., Hart, D.M. & Cohen, P.R. Designing agents to plan and act in their environments. Abstract for *The Workshop on Automated Planning for Complex Domains* at the Eighth National Conference on Artificial Intelligence. Boston, MA, 1990.

Howe, Adele E. Integrating adaptation with planning to improve behavior in unpredictable environments. In *Planning in Uncertain, Unpredictable, or Changing Environments*. *Working Notes of the 1990 AAAI Spring Symposium*. Also, Technical Research Report #90-45, Systems Research Center, University of Maryland, 1990.

Silvey, P.E., Loisel, C.L. & Cohen, P.R. Intelligent data analysis. *Working Notes of the AAAI-92 Fall Symposium on Intelligent Scientific Computation*. Cambridge, MA. October 23-24, 1992.

### 2.2.5. Books or Parts Thereof Published

Cohen, P.R. *Empirical Methods for Artificial Intelligence*. Textbook in preparation.

Cohen, P.R. *Architectures for Reasoning Under Uncertainty*. 1990. Readings in Uncertain Reasoning. Glenn Shafer and Judea Pearl, Eds., Morgan-Kaufmann.

Howe, Adele E. and Cohen, Paul R. How evaluation guides AI research. Reprinted in *A Sourcebook of Applied Artificial Intelligence*. Gerald Hopple and Stephen Andriole, Eds. TAB Books, Inc. 1990. (Originally published in *AI Magazine*, Winter, 1988.)

### 2.2.6. Ph.D. Dissertations

Howe, A.E. *Accepting the Inevitable: The Role of Failure Recovery in the Design of Planners*. February, 1993.

### 2.2.7. Unrefereed Reports and Articles

Anderson, S.D. & Hart, D.M. Monitoring interval. *EKSL Memo #11*. Experimental Knowledge Systems Laboratory, Dept. of Computer Science, Univ. of Massachusetts, Amherst. 1990.

Cohen, P.R., Hart, D.M., & Devadoss, J.K. Models and experiments to probe the factors that affect plan completion times for multiple fires in Phoenix. *EKSL Memo #17*. Experimental Knowledge Systems Laboratory, Dept. of Computer Science, Univ. of Massachusetts, Amherst. 1990.

Fisher, D.E. Common Lisp Analytical Statistics Package (CLASP): User manual. *Technical Report 90-85*, Dept. of Computer Science, Univ. of Massachusetts, Amherst. Revised and expanded, 1991.

Greenberg, M.L. & Westbrook, D.L. The Phoenix testbed. *Technical Report 90-19*, Dept. of Computer Science, Univ. of Massachusetts, Amherst, MA, 1990.

Hansen, E.A. & Cohen, P.R. Learning a decision rule for monitoring tasks with deadlines. *Technical Report #92-80*, Dept. of Computer Science, Univ. of Massachusetts, Amherst. 1992.

Hansen, E.A. The effect of wind on fire spread. *EKSL Memo #10*. Experimental Knowledge Systems Laboratory, Dept. of Computer Science, Univ. of Massachusetts, Amherst. 1990.

Hansen, E.A. A model for wind in Phoenix. *EKSL Memo #12*. Experimental Knowledge Systems Laboratory, Dept. of Computer Science, Univ. of Massachusetts, Amherst. 1990.

Howe, A.E. & Cohen, P.R. Debugging plan failures by analyzing execution traces. *Technical Report #92-22*, Dept. of Computer Science, Univ. of Massachusetts, Amherst. 1992.

Howe, A.E. Did we measure what we thought?: Problems with the method cost measure. *EKSL Memo #16*. Experimental Knowledge Systems Laboratory, Dept. of Computer Science, Univ. of Massachusetts at Amherst. 1991.

Silvey, P.E. Phoenix baseline fire spread models. *EKSL Memo #13*. Experimental Knowledge Systems Laboratory, Dept of Computer Science, Univ. of Massachusetts, Amherst. 1990.

## 2.3. Conferences, Workshops and Presentations

### 2.3.1. Invited Presentations

Cohen, P.R.

- Member of a panel entitled "Planning under uncertainty" at the *AAAI Workshop on Production Planning, Scheduling and Control*, which focused on scheduling strategies for managing uncertainty in complex, real-time environments, July 1992.
- Invited presentation on EKSL's current research at a two day meeting of the Institute for Defense Analysis's Information and Science Technology Advisory Group on Simulation in Washington, DC, June 1992.
- Member of a panel entitled "The empirical evaluation of planning systems: Promises and pitfalls" at the *First International Conference on AI Planning Systems* at the Univ. of Maryland, June 1992.
- Methods for agentology: General concerns, specific examples. Invited talks at Virginia Polytechnic Institute and the Univ. of West Virginia. April 1992.
- Three examples of statistical modeling of an AI program. Invited talk at the Univ. of Texas, Austin. March 1992.
- Member of a panel entitled "The future of expert systems" chaired by Dr. Y.T. Chen of NSF at the World Congress on Expert Systems, December 1991, in Orlando, Florida.
- A brief report on a survey of AAAI-90, some methodological conclusions, and an example of the MAD methodology in Phoenix. Keynote address, *AAAI Spring Symposium on Implemented AI Systems*. Palo Alto, CA. March, 1991.
- Methodological problems, a model-based design and analysis methodology, and an example. Keynote address at the *International Symposium on Methodologies for Intelligent Systems*. Knoxville, TN. October 1990.
- What is an interesting environment for AI planning research? Panel moderator, *Workshop on Automated Planning for Complex Domains* at the *Eighth National Conference on Artificial Intelligence*. Boston, MA. July 1990.
- Modeling for AI system design. Imperial Cancer Research Fund, London, England. June 1990.
- Modelling for AI system design. Digital Equipment Corporation, Galway, Ireland. June 1990.
- Fire will destroy the pestilence, or, How natural environments will drive out bad methodology. Texas Instruments, Dallas. May 1990.
- Designing autonomous agents. Thayer School of Engineering, Dartmouth College. November 1989.

Howe, A.E.

"Accepting the Inevitable: The Role of Failure Recovery in the Design of Planners".

- Dept. of Computer Science, Oregon State Univ., March 1992.
- Dept. of Computer Sciences, Purdue Univ., March 1992.
- Computer Science Dept., Univ. of Maryland, Baltimore County, March 1992.
- Computer Science Dept., Colorado State Univ., April 1992.
- Dept. of Electrical and Computer Engineering, Clarkson Univ., April 1992.
- School of Computer Science, Carnegie Mellon Univ., April 1992.

### 2.3.2. Contributed Presentations

Cohen, P.R.

- Welcoming address (untitled) at the NSF/DARPA *Workshop on Artificial Intelligence Methodology*. Northampton, MA. June 1991.
- The Phoenix project: Responding to environmental change. *Workshop on Innovative Approaches to Planning, Scheduling, and Control*. San Diego, CA. November 1990.
- Intelligent real-time problem solving: Issues and examples. Presented at the Intelligent Real-Time Problem Solving Workshop, Santa Cruz, CA, November 1989.

Hart, D.M.

- Predicting and explaining success and task duration in the phoenix planner. Paper presentation at the *First International Conference on AI Planning Systems* at the University of Maryland, June 1992.

Howe, A.E.

- Detecting and explaining dependencies. Poster presented at the *Fourth International Workshop on AI and Statistics*, Ft. Lauderdale, FL. January 1993.
- Analyzing failure recovery to improve planner design. Paper presented at the *Tenth National Conference on Artificial Intelligence*. San Jose, CA. July 1992.
- Isolating dependencies on failure by analyzing execution traces. Poster presentation at the *First International Conference on AI Planning Systems* at the University of Maryland, June 1992.
- Failure recovery analysis as a tool for plan debugging. *AAAI Spring Symposium on Computational Considerations in Supporting Incremental Modification*. Palo Alto, CA. March 1992.
- Failure recovery: A model and experiments. Paper presented at the *Ninth National Conference on Artificial Intelligence*. Pasadena, CA. July 1991.
- Adaptable planning in the Phoenix system. Poster presentation at the *Symposium on Learning Methods for Planning and Scheduling*, Palo Alto, CA. January 1991.
- Designing agents to plan and act in their environments. *Workshop on Automated Planning for Complex Domains* at the *Eighth National Conference on Artificial Intelligence*. Boston, MA. July 1990.
- Integrating adaptation with planning to improve behavior in unpredictable environments. *Planning in Uncertain, Unpredictable, or Changing Environments*, AAAI Spring Symposium, Palo Alto, CA, March 1990.

Powell, G.M.

- Operational planning and monitoring with envelopes. *IEEE Fifth Annual AI Systems in Government Conference*. Washington, DC. May 1990.

### 2.3.3. Tutorials

Cohen, P.R.

- Offered a tutorial with Bruce Porter (of the University of Texas) at the *Tenth National Conference on Artificial Intelligence* entitled "Experimental Methods in Artificial Intelligence." This tutorial used several examples from our research under this contract, including the results reported in (Hart & Cohen 1992) and (Cohen, St. Amant & Hart, 1992). This tutorial will be offered again at the *Eleventh National Conference on Artificial Intelligence* in 1993.

Hart, D.M.

- Tutorial on the Phoenix Testbed and real-time research being conducted in Phoenix at Wright Patterson AFB, April, 1991. This tutorial was offered for potential consumers of IRTPS research results.

## 2.4. Awards, Promotions, Honors

Cohen, P.R.

- Elected a Fellow of the American Association for Artificial Intelligence.
- Elected a Councilor of the American Association for Artificial Intelligence for the term 1991-94.
- Appointed to the Information and Science Technology Advisory Group on Simulation, Institute for Defense Analysis.
- As an AAAI Councilor, served as Chair of the AAAI-93 Tutorial Committee, Co-chair of the 1992-93 Symposium Committee, and Assistant to the Chair for the Program Committee of AAAI-93.
- Chairman, NSF/DARPA Workshop on AI Methodology. University of Massachusetts. June, 1991
- Organizing Committee, AAAI Workshop on Intelligent Real-Time Problem Solving. Anaheim, CA. July, 1991.
- Program Committee, Sixth International Symposium on Methodologies for Intelligent Systems (ISMIS'91). Charlotte, NC. October 1991.

Hansen, E.A.

- Recipient of an ARPA/AFOSR Augmentation Award for Science and Engineering Research Training for an investigation of monitoring strategies related to EKSL work in pathology detection in Phoenix and in transportation planning.

Adele E. Howe

- Appointed Assistant Professor of Computer Science at Colorado State University, September, 1992.



## 2.5 Technology Transfer

### 1992: DARPA/Rome Labs Planning Initiative

Much of the work we have done in Phoenix is being transferred to the DARPA/Rome Labs Planning Initiative (PI). This includes the creation of a testbed environment for controlled experimentation, our ongoing work with envelopes and monitoring, and the use of path analysis to build causal models of program behavior. Part of the PI effort involves building a Common Prototyping Environment (CPE) for integrating and evaluating components of the evolving planning and scheduling architecture. The CPE will have many of the kinds of testbed features we built into Phoenix and are building into our simulation of the transportation planning domain.

**Using Simulation Testbeds to Design AI Planners.** Over the last four years, with funding from DARPA, URI, and IRTPS, we have created a testbed environment for Phoenix. This effort included instrumenting the system, baselining the simulated environment, and providing such facilities as predefined scenarios, scripts, and primitives for experiment definition and data collection. We integrated the first version of CLASP<sup>3</sup> into this testbed environment to provide built-in data analysis capabilities. Since that time we have extended many of these testbed features in Phoenix and ported them to our simulation of the sea transport domain in the PI. Many of these features will soon find their way into the CPE under the direction of the Issues Working Group on Prototyping Environment, Instrumentation and Methodology. Paul Cohen is the co-chair of this group along with Mark Burstein of BBN.

**Envelopes and Monitoring.** Our work with envelopes and monitoring in Phoenix is directly applicable to the design of *pathology demons* for the transportation planning problem that is the domain of the PI. Pathology demons are designed to detect typical pathologies that arise during the execution of large-scale transportation plans and to help the user (interacting through informed visualizations) steer the plan around the pathologies. Envelope-like representations of plan progress tell us whether we are keeping trim to the schedule, and our developing theories of appropriate monitoring strategies tell us how often to monitor and what to watch.

**Building Causal Models of Program Behavior using Path Analysis.** During this contract we began applying path analysis to our work in Phoenix. We think path analysis will provide a powerful technique for building causal models from large data sets such as those generated by experiments in Phoenix and the PI's CPE. We recently enhanced CLASP by adding a module for path analysis. Using a graphical interface, the user draws a directed graph of hypothesized causal influences among independent and dependent variables, and the path analysis module calculates the corresponding path coefficients (strengths of influence) along the arcs (one per arc). The user can explore variations on the model simply by modifying nodes and arcs in

---

<sup>3</sup> The Common Lisp Analytical Statistics Package (CLASP) was originally implemented on the TI Explorer for analyzing Phoenix experiments. For more on CLASP, see Fisher (1990).

the graph -- recalculation is done automatically. Such a facility will help users fit causal models to planner behaviors that arise in CPE simulations. As part of a new contract in the PI, we will be building an experiment module that automatically generates causal models from data sets to aid developers in the design and evaluation of AI planning systems.

## 1991

In April, David Hart (EKSL Lab Manager) and David Westbrook (EKSL Systems Developer) visited Mark Burstein at BBN to see a demonstration of the Dynamic Replanning and Analysis Tool (DART) being developed as part of the DARPA Planning Initiative for USTRANSCOM. As part of this initiative, EKSL began work under contract in the last quarter of FY91. Our visit to BBN was designed as an exchange of information about the use of simulation in complex planning problems.

A significant part of BBN's contribution to DART is the Prototype Feasibility Estimator (PFE), a dispatch scheduling program designed to demonstrate the gross feasibility of USTRANSCOM operation plans. These plans are currently developed by USTRANSCOM planners, but will eventually be generated by planning technology produced by this initiative. We discussed possibilities for enhancing PFE to simulate the movement of resources and cargo through the transportation network, much as we simulate the movement of fire-fighting agents in the Phoenix world. Such a simulation could be used to watch a plan execute, allowing operators to recognize problems as they develop and "steer" the plan around them.

Mark Burstein visited EKSL in August to see Phoenix and continue the discussions mentioned above. While here he consulted on our efforts to get PFE running and gave us some invaluable hands-on assistance.

## 1990

Paul Cohen and David Hart visited the Decision Systems Laboratory at Texas Instruments in Dallas, May 24-25. Cohen presented a talk entitled "Fire will Destroy the Pestilence, or, How Natural Environments will Drive Out Bad Methodology." Phoenix was demonstrated for the DSL, and we looked at a number of their projects, including CACTUS, a battlefield planning system that is conceptually similar to Phoenix, but implemented differently. We discussed doing a comparative analysis of these two systems to show they fall within an equivalence class with respect to the task environments and design of agents for those environments. Such an analysis would attempt to show that both systems can be represented using the same underlying model for the task environment and agent design, thus substantiating the methodological approach we advocate.

We also discussed at length with TI the use of visualization techniques to aid in the interpretation and analysis of a system that simulates shop floor activities in a semiconductor fabrication plant. The simulation allows experimentation with various scheduling strategies to improve plant throughput. However, the volume of data it

produces overwhelms the capabilities of traditional data analysis techniques. Our discussions focused on ways of visualizing pathologies that arise during (the simulation of) shop floor processing that cause the operant scheduling strategy to perform poorly or fail, so that the user can intervene as problems develop and explore the causes by pausing and interacting with the system graphically. These ideas are based on our work in Phoenix with simulation, graphical interfaces, and envelopes; they are also the subject of another DARPA contract (*Visualization and Modeling for Interactive Plan Development and Plan Steering*).

Paul Cohen presented a talk entitled "Modelling for AI System Design" at Digital Equipment Corporation in Galway, Ireland, and at the Imperial Cancer Research Fund in London on June 25. These talks led to plans to hold a workshop sponsored by NSF and DARPA in early 1991 on methodology in AI research. DEC considered using the Phoenix planning system as part of a market simulator for new computer products, designed to allow DEC marketing executives to simulate alternative pricing structures for the products in order to find the most advantageous. Phoenix planning agents play the roles of DEC's competitors, responding to the introduction of DEC products with changes in their own product lines and pricing structures.

Gerald M. Powell was a visiting faculty member at EKSL under the Secretary of the Army Research and Study Fellowship Program. Dr. Powell, who works for the Center for Command, Control, and Communications Systems, CECOM, Ft. Monmouth, New Jersey, had investigated computational approaches to various problems in battlefield planning for the previous five years, and was very interested in the design and development of Phoenix. He had worked previously with Paul Cohen applying envelopes to an operations planning problem in battlefield management. During the reporting year he studied (among other issues) the application of approximate processing techniques for real-time control in Phoenix.

## 2.6 Software Prototypes

### 1991

In 1991 we enhanced the Common Lisp Analytical Statistical Package (CLASP) by adding several new statistical tests and porting it to a UNIX-based Common Lisp environment. CLASP was developed (under URI funding) for the statistical analysis of large data sets on the TI Explorer. This modularized system is the kernel of the Phoenix experimental interface. It can be used as an interactive analysis tool for data generated experimentally, providing powerful data manipulation tools, standard statistical tests, and plotting capabilities. In addition, it can be accessed as a runtime library by programs (e.g., Phoenix agents) using statistical and probabilistic models.

This ported version of CLASP will be integrated into the Common Prototyping Environment (CPE) being developed at BBN for the DARPA Planning Initiative (see Technology Transfer), where it will be used to analyze the dynamics of the transportation problem, as well as the planning/scheduling techniques applied to the problem (now completed, 1993). While the interface being developed for this ported version is specific to CPE, future plans include providing a generic CLASP interface using the Common Lisp Interface Manager (now completed, 1993). This version of CLASP would run standalone in most Common Lisp environments. To support such an implementation, we have developed and documented an automated test suite for the CLASP package that validates its functionality. This test suite can be run to uncover bugs and inconsistencies between systems and versions whenever CLASP is ported to a new platform.

### 1990

We have made Phoenix available as an instrumented testbed for use by other researchers designing autonomous agents for complex, real-time environments as part of the Intelligent Real-time Problem Solving initiative (Cohen, Howe & Hart 1989) and as part of a new initiative in evaluation and benchmarking of planning systems for complex, dynamic environments (Cohen & Howe 1990; Hart & Cohen 1990). It is also being used by the Cooperative Distributed Problem Solving Laboratory (under Victor Lesser) at the University of Massachusetts (Moehlman & Lesser 1990).

## 2.7 References

- Anderson, S.D., Hart, D.M., & Cohen, P.R. 1991. Two ways to act. *AAAI Spring Symposium on Integrated Intelligent Architectures*. Published in the *SIGART Bulletin*, Vol. 2, No.4, pp. 20-24.
- Anderson, S.D. & Hart, D.M. 1990. Monitoring interval. *EKSL Memo #11*. Experimental Knowledge Systems Laboratory, Dept. of Computer Science, Univ. of Massachusetts, Amherst.
- Asher, H.B. 1983. *Causal Modeling*. Sage Publications.
- Barto, A.G.; Sutton, R.S.; and Watkins, C.J.C.H., 1990. Learning and sequential decision making. In *Learning and Computational Neuroscience: Foundations of Adaptive Networks*. M. Gabriel and J. W. Moore (Eds.), MIT Press, Cambridge, MA. Pp. 539-602.
- Cohen, P.R. *Empirical Methods for Artificial Intelligence*. Textbook in preparation.
- Cohen, P.R., St. Amant, R. & Hart, D.M., 1992. Early warning of plan failure. false positives and envelopes: Experiments and a model. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates, Inc. Pp. 773-778.
- Cohen, P.R., 1991a. Timing is everything. *EKSL Memo #21*. Experimental Knowledge Systems Laboratory, Computer Science Dept., Univ. of Massachusetts, Amherst.
- Cohen, P.R. 1991b. A Survey of the Eighth National Conference on Artificial Intelligence: Pulling together or pulling apart? *AI Magazine*, 12(1), 16-41.
- Cohen, P.R. 1990. Designing and Analysing Strategies for Phoenix from Models. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, Katia Sycara (Ed.). Morgan-Kaufman. Pp. 9-21.
- Cohen, P.R. & Howe, A.E. 1990. Benchmarks are not enough; Evaluation metrics depend on the hypothesis. *Collected Notes from the Benchmarks and Metrics Workshop*. Technical Report FIA-91-06, NASA Ames Research Center. Pp. 18-19.
- Cohen, P.R., Hart, D.M., & Devadoss, J.K. 1990. Models and experiments to probe the factors that affect plan completion times for multiple fires in Phoenix. *EKSL Memo #17*. Experimental Knowledge Systems Laboratory, Dept. of Computer Science, Univ. of Massachusetts, Amherst.
- Cohen, P.R., Howe, A.E. & Hart, D.M. 1989. Intelligent real-time problem solving: Issues and examples. *Intelligent Real-Time Problem Solving: Workshop Report*, edited by Lee D. Erman, Santa Cruz, CA.
- Cohen, P.R., Greenberg, M.L., Hart, D.M. & Howe, A.E., 1989. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3): 32-48.
- Fikes, R., Hart, P. & Nilsson, N., 1972. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4), Pp. 251-288.
- Fikes, R., 1971. Monitored execution of robot plans produced by STRIPS. *Proceedings of the IFIP Congress, 1971*, Ljubljana, Yugoslavia. Pp. 189-194.
- Fisher, D.E. 1990. Common Lisp Analytical Statistics Package (CLASP): User manual. *Technical Report 90-85*, Dept. of Computer Science, Univ. of Massachusetts, Amherst. Revised and expanded, 1991.

- Haben, D. & Shin, K., 1990. Application of real-time monitoring to scheduling tasks with random execution times. *IEEE Transactions on Software Engineering*, Vol. 16, No. 12, Pp. 1374-1389.
- Hansen, E.A. 1993. Monitoring as a sequential decision problem. Master's thesis in preparation.
- Hansen, E.A. & Cohen, P.R. 1992a. Learning a decision rule for monitoring tasks with deadlines. Computer Science Technical Report 92-80. Univ. of Massachusetts, Amherst.
- Hansen, E.A. & Cohen, P.R. 1992b. Monitoring plan execution: A survey. In preparation for *AI Magazine*.
- Hansen, E.A. 1990a. The effect of wind on fire spread. *EKSL Memo #10*. Experimental Knowledge Systems Laboratory, Dept. of Computer Science, Univ. of Massachusetts, Amherst.
- Hansen, E.A. 1990b. A model for wind in Phoenix. *EKSL Memo #12*. Experimental Knowledge Systems Laboratory, Dept. of Computer Science, Univ. of Massachusetts, Amherst.
- Hart, D.M. & Cohen, P.R., 1992. Predicting and explaining success and task duration in the Phoenix planner. *Proceedings of the First International Conference on AI Planning Systems*. Morgan Kaufmann. Pp. 106-115.
- Hart, David M. and Cohen, Paul R. 1990. Phoenix: A testbed for shared planning research. *Collected Notes from the Benchmarks and Metrics Workshop*. Technical Report FIA-91-06, NASA Ames Research Center. Pp. 20-27.
- Hart, D.M., Anderson, S.D., & Cohen, P.R. 1990. Envelopes as a vehicle for improving the efficiency of plan execution. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, Katia Sycara (Ed.). Morgan-Kaufman. Pp. 71-76.
- Howe, A.E. 1993. *Accepting the Inevitable: The Role of Failure Recovery in the Design of Planners*. PhD. thesis. Dept. of Computer Science. Univ. of Massachusetts, Amherst.
- Howe, A.E., 1992. Analyzing failure recovery to improve planner design. *Proceedings of the Tenth National Conference on Artificial Intelligence*. AAAI Press/MIT Press. Pp. 387-392.
- Howe, A.E. & Cohen, P.R. 1991. Failure recovery: A model and experiments. *Proceedings of the Ninth National Conference on Artificial Intelligence*. AAAI Press/MIT Press. Pp. 801-808.
- Howe, A.E. & Cohen, P.R. 1990. Responding to environmental change. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, Katia Sycara (Ed.). Morgan-Kaufman. Pp. 85-92.
- Lesser, V.R., Pavlin, J. & Durfee, E., 1988. Approximate processing in real-time problem solving. *AI Magazine*, 9(2): 49-61.
- Li, C.C., 1975. *Path Analysis-A Primer*. Boxwood Press.
- Moehlman, T. & Lesser, V. 1990. Cooperative planning and decentralized negotiation in multi-fireboss Phoenix. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, Katia Sycara (Ed.). Morgan-Kaufman. Pp. 144-159.
- Silvey, P.E. 1990. Phoenix baseline fire spread models. *EKSL Memo #13*. Experimental Knowledge Systems Laboratory, Dept of Computer Science, Univ. of Massachusetts, Amherst.

### 3. Failure Recovery and Failure Recovery Analysis in Phoenix

This paper, summarizing part of Adele E. Howe's thesis "Accepting the inevitable: The role of failure recovery in the design of planners," has been submitted to the IEEE Transactions on Knowledge and Data Engineering. Some of these results appeared previously in the Proceedings of the Ninth and Tenth National Conference on Artificial Intelligence (1991 and 1992).

## Improving the Reliability of AI Planning Systems by Analyzing their Failure Recovery\*

Adele E. Howe  
Computer Science Department  
Colorado State University  
Fort Collins, CO 80523

email: howe@cs.colostate.edu  
telephone: 303-491-7589

Keywords: AI. Planning. failure recovery. reliability. debugging

March 31, 1993

### Abstract

As planning technology improves, AI planners are being embedded in increasingly complicated environments: ones that are particularly challenging even for human experts. Consequently, failure is becoming both increasingly likely for these systems (due to the difficult and dynamic nature of the new environments) and increasingly important to address (due to the systems' potential use on real world applications). This paper describes the development of a failure recovery component for a planner in a complex simulated environment and a procedure (called Failure Recovery Analysis) for assisting programmers in debugging that planner. The failure recovery design is iteratively enhanced and evaluated in a series of experiments. Failure Recovery Analysis is described and demonstrated on an example from the Phoenix planner. The primary advantage of these approaches over existing approaches is that they are based on only a weak model of the planner and its environment, making them suitable when the planner is being developed and making them easily generalizable to other planners and environments. Together, failure recovery and Failure Recovery Analysis improve the reliability of the planner by preventing failures due to bugs in the planner and repairing failures during execution.

---

\*This research was supported by a DARPA-AFOSR contract F49620-89-C-00113, the National Science Foundation under an Issues in Real-Time Computing grant, CDA-8922572, and a grant from the Office of Naval Research under the University Research Initiative N00014-86-K-0764. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. This research was conducted as part of my PhD thesis research at the University of Massachusetts. I would like to thank my thesis advisor, Paul Cohen, for his advice, guidance and supervision of this research.

## 1 Introduction

As planning technology improves, AI planners are being embedded in increasingly complicated environments: ones that are particularly challenging even for human experts. Consequently, failure is becoming both increasingly likely for these systems and increasingly important to address. Failure is increasingly likely because of the difficult and dynamic nature of the new environments; failure is increasingly important to address because of the systems' potential use on applications such as scheduling manufacturing production lines [25] and Hubble space telescope time [16], and controlling robots [15].

AI planners determine a course of action: it may be the next action to be taken or may be a long sequence of actions. Plan failures may be caused by actions not having their intended effects, by unexpected environmental changes or by inadequacies in the planner itself. This paper describes an integrated approach to dealing with failure, which uses feedback from failure recovery to help debug plan failures. This approach was developed as part of my thesis research [9] on improving the reliability of the Phoenix planner.

### 1.1 Approaches to Improving Planner Reliability

In general, software failures have been addressed in two ways: automated failure recovery and debugging. The first involves designing the software to detect and repair its own failures. The second method is to debug the software to remove the causes of failure.

The first part of this paper describes the design and the methodology behind the design of automated failure recovery for an AI planner. The second part describes a procedure called *Failure Recovery Analysis* (or FRA) for analyzing the performance of failure recovery and determining how the planner's knowledge base might influence the occurrence of particular failures. The two parts are tightly related: as Figure 1 shows, failure recovery is like a loop within FRA.

Failure recovery repairs failures that arise during normal planning and acting. FRA "watches" failure recovery for clues to bugs in the planner and informs the designer of the bugs. The designer can then attempt to repair the plan knowledge base to prevent the failure from occurring again and, using FRA, can evaluate whether the repair was successful.

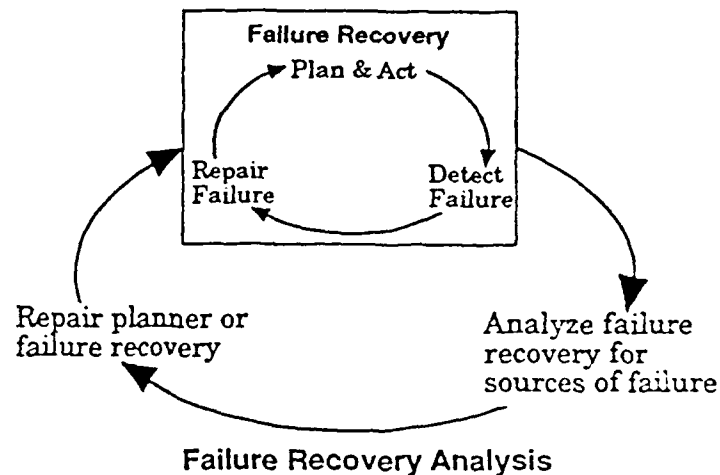


Figure 1: Relationship of failure recovery to Failure Recovery Analysis

## 1.2 The Target Planner and its Environment

The Phoenix system [4] serves as the laboratory for this research in failure recovery. As shown in Figure 2, Phoenix provides the simulated environment, a planner with a set of plan knowledge bases for agents, and an experimental interface for automatically controlling experiments and collecting data. Its environment is forest fire fighting in Yellowstone National Park.

The goal of forest fire fighting is to contain fires as efficiently as possible. Forest fires spread in irregular shapes, at variable rates, as a function of ground cover, elevation, moisture content, wind speed and direction, and natural boundaries (e.g., bodies of water and large roads). Fires are contained by removing fuel from their paths, causing them to burn out. This process, called

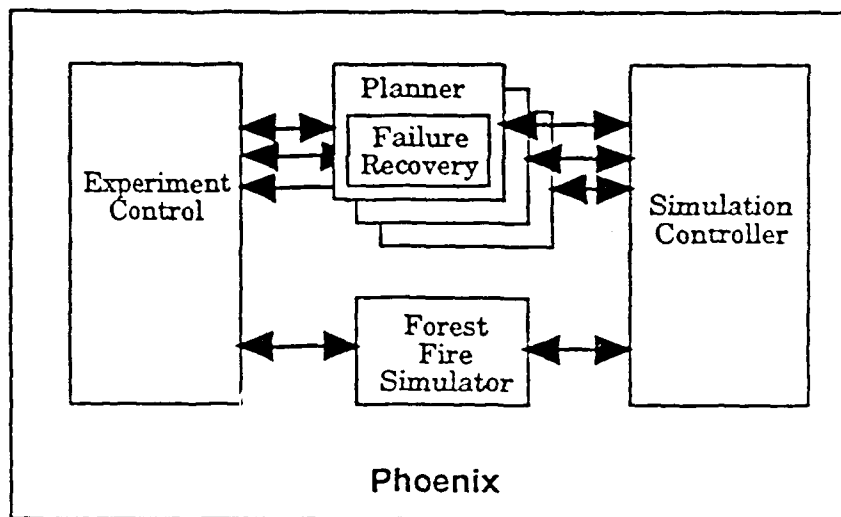


Figure 2: Diagram of the separate processes that comprise the Phoenix system. The arcs between the processes indicate a transfer of data between processes.

building fireline, requires the coordination of many agents to surround the fire with fireline or natural boundaries. One agent, the fireboss, coordinates the activities of field agents, bulldozers, to surround the fire with fireline. Other agents, watchtowers, gasoline carriers, and helicopters, support the activities of the fireboss and bulldozers by gathering information and delivering gasoline to refuel agents in the field.

Figure 3 shows the interface to the simulator. The map in the upper part of the display depicts Yellowstone National Park north of Yellowstone Lake. Features such as wind speed and direction are shown in the window in the upper left, and geographic features such as rivers, roads and terrain types are shown as light lines or grey shaded areas. Four bulldozers are building fireline around a fire near the center of the figure. A watchtower is visible at the top near the center.

All Phoenix agents have the same agent architecture, which consists of sensors, effectors, reflexes and a planner. Sensors perceive the state of the environment local to the agent, and effectors change the environment. Together, reflexes and the planner form a two-layer control

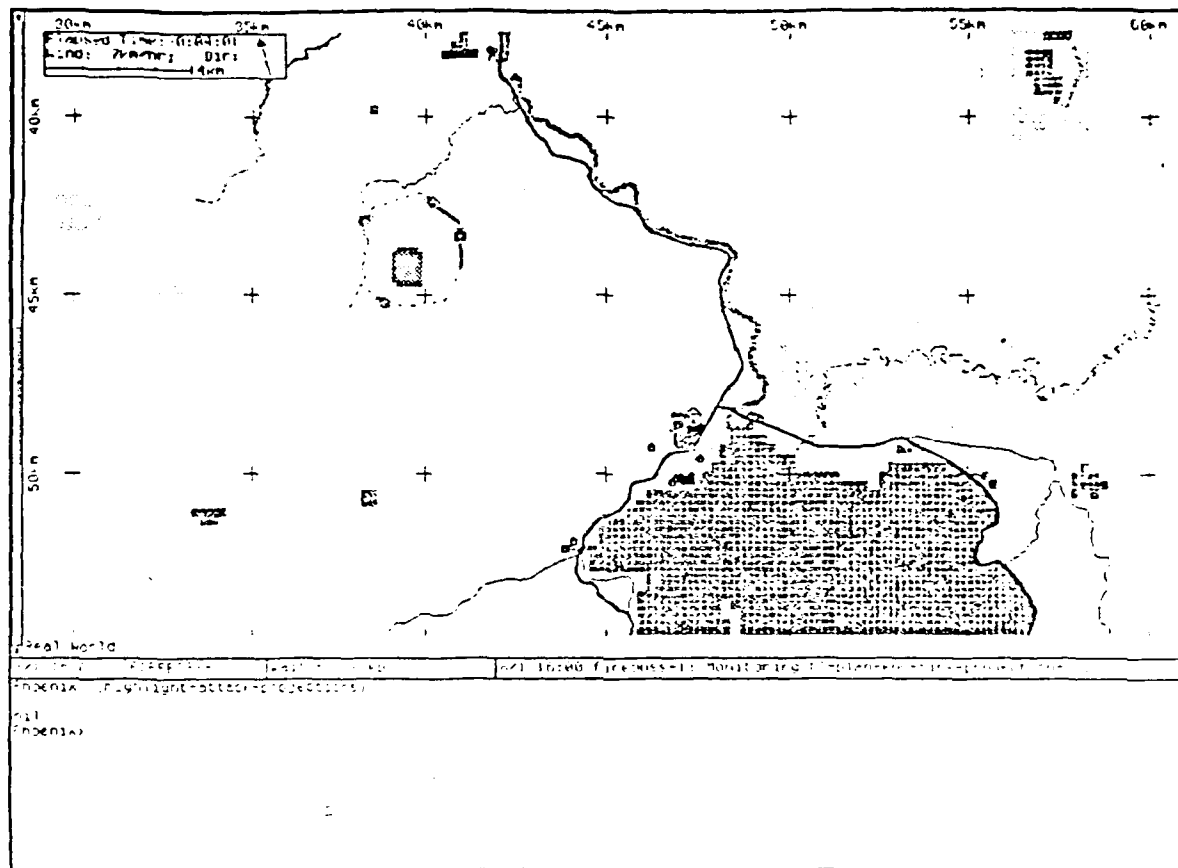


Figure 3: View from Phoenix Simulator of bulldozers fighting a fire

system. Each layer provides a particular level of competence. Reflexes address changes that occur faster than the cognitive component can respond, and the planner coordinates actions and avoids detrimental plan interactions.

Phoenix plans fail for lots of reasons. The environment can change unpredictably, so if the planner bases a plan on slow, different or no change in the environment, the plan can fail. Phoenix agents are limited in their abilities to sense the environment, so plans can fail because they are based on obsolete or uncertain information. Phoenix plans also fail because they include bugs and have not been tested in all possible situations.

## 2 Failure Recovery

The purpose of failure recovery is to repair, as efficiently as possible, the plan so as to resume progress toward the failed plan's goal. Several areas of AI have developed automated failure recovery techniques for knowledge based systems (most notably: robotics [2,6,26], and planning [7,23,27,29]). In essence, the different approaches all have the same basic steps, as shown in Figure 4. The system continues to execute (planning and acting) until it recognizes that its actions are failing. Then, the system deals with the failure by taking some corrective action.

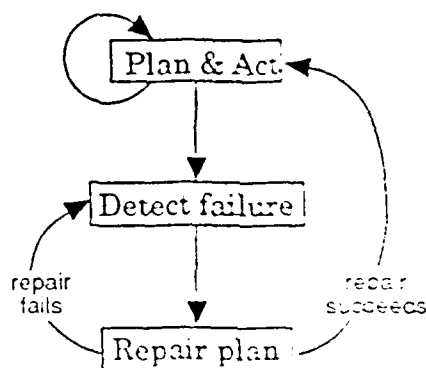


Figure 4: Flow of control between planning/acting and recovering from failures. Normal activity resumes after a failure has been repaired.

The two basic approaches to failure recovery are backward and forward recovery [14]. In backward recovery, the system is returned to some previous correct state and resumes execution from there. Backward recovery requires that actions can be undone and that the system has full control over its environment; these requirements preclude backward recovery for most robotics and planning systems. Forward recovery transforms the failure state to a correct state by repairing the failure [6]. Approaches to forward recovery differ on how to accomplish the appropriate transformation or repair through formal, heuristic, or informal planning or learning.

Formal analysis approaches decide how to repair failures by reference to a complete model

of failures and their causes. Time Petri net models, real-time logic and fault tree analysis are common techniques for modeling the causes of failures and for guiding recovery in software (e.g., [13,18]). Formal theories of planning and replanning have been proposed to guide plan repair (e.g., [11,17]).

Failure recovery can be treated as a normal part of the planning process. Lesser's Functionally Accurate, Cooperative (FA/C) paradigm for distributed problem solving [12] and Ambros-Ingerson et al.'s IPDM [1] treat failure recovery as just another planning or problem solving task.

Both recovery through formal analysis and as part of planning require that the planner employs a strong model of what to do in any situation, including failures. Heuristic approaches allow for gaps in knowledge and apply recovery methods to repair the failure. Typically, heuristic approaches operate by "retrieve-and-apply" [20] which maps observed failures to suggested responses. The most comprehensive and commonly used domain-independent strategy is replanning (e.g., [14,29]), which involves restating the planning problem and re-initiating planning. Other domain-independent recovery methods are based on an informal model of how that planner's plans fail and can be modified (e.g., [3,29]). Robotics and other areas that treat planning as a subtask have favored domain-dependent recovery methods (e.g., [15,28]).

Many different approaches have been proposed for failure recovery: most depend either on the domain or on the planner design. However, the literature offers few suggestions about how to design failure recovery for a *new* planner or domain. For constructing domain-dependent recovery programs, Nof et al. [19] propose a four-step framework: analyze the task, develop alternative recovery strategies, determine a selection strategy, and update based on experience with the system. Similarly, Simmons advocates starting the system with basic competence at its task (i.e., no failure recovery) and then adding execution monitoring and failure recovery methods as needed [22]. Wilkins [29] advocates combining both domain-independent and

domain-specific methods; the system can try the more efficient domain-specific methods when they are available, but fall back on the domain-independent, when necessary.

## 2.1 Designing Failure Recovery for Phoenix

Most of the previously mentioned approaches to failure recovery classify the failure and select from a set of methods for adapting the plan in progress. The approaches differ in their classification of failures and their recovery methods: so how does a designer decide on a failure classification and a set of recovery methods for a new domain? The approach adopted in this research is to begin with a flexible method selection mechanism and a core set of recovery methods and then refine the set by evaluating failure recovery performance in the host environment. This section will describe the basic design of failure recovery for the Phoenix planner and the methodology that directed the re-design and evaluation of that failure recovery component.

**Design of Failure Recovery:** In Phoenix, failures can be detected during construction of the plan or during its execution, and can be due to anything from rapid change in the environment to bugs in the plans. At present, the Phoenix fireboss detects ten types of failures, and bulldozers detect five types, as shown in Table 1. The classifications of failure types are largely domain-dependent. Failures are classified by what is known about what finally blocked the plan from continuing successfully; the actual cause of the failure is unknown.

In Phoenix, failure recovery is initiated in response to detecting a failure, an event that precludes successful completion of some plan. Failure recovery iteratively tries recovery methods until one works, at which point the plan is resumed. For example, Figure 5 depicts the flow of control between failure recovery and the rest of the plan for an *insufficient progress* failure. A failure detection mechanism determined that the plan is taking too long to complete. Failure recovery deals with this failure by searching a library of recovery methods for those applicable

Table 1: List of Failure Types for the Phoenix Fireboss and Bulldozers

Agent	Failure Types
Fireboss	(CCP) Can't Calculate Path (CCV) Can't Calculate Variable (CFP) Can't Find Plan (FNE) Fire Not Encircled when it should be (IP) Insufficient Progress to contain the fire (NER) Not Enough Resources to contain the fire (NRS) No Remaining fireline Segments to build (PRJ) Can't Calculate Projection of fire (PTR) Can't Calculate Path to Road (RU) Resource Unavailable (RU)
Bulldozer	(CCP) Can't Calculate Path (DOP) Deadly Object in Path (XVV) No Variable Value (OOF) Out Of Fuel (PM) Position Mismatch

to the failure type. It selects one method from the possible set and executes it. If the recovery method succeeds, then failure recovery completes and the rest of the plan executes; otherwise, it abandons the current attempt, selects another method and tries again. This process continues until a method succeeds or it runs out of methods to try.

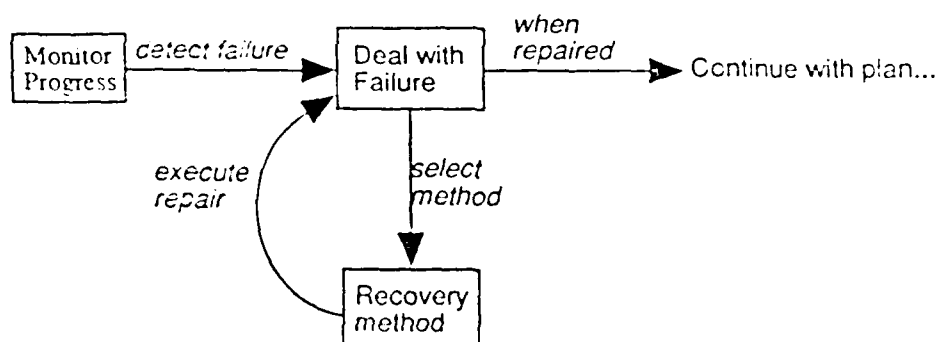


Figure 5: Abstracted view of the flow of control between failure recovery and the rest of the plan.

The recovery methods make mostly simple repairs to the structure of the failed plan. Consistent

Table 2: Set of Recovery Methods for Phoenix

Method	Description
WATA	Wait and try the failed action again.
RV	Re-calculate one variable used in the failed action.
RA	Re-calculate all variables used in the failed action.
SA	Substitute a similar plan step for the failed action.
RP	Abort the current plan and re-plan at the parent level (i.e., the level in the plan immediately above this one).
RT	Abort current plan and re-plan at the top level (i.e., redo the entire plan).

quently, these methods can be used in different situations, do not require expensive explanation, and provide a response to any failure. This strategy sacrifices efficiency for generality and results in a planner capable of responding to most failures, but perhaps in a less than optimal manner. The six methods in the basic library are listed in Table 2.

The first four methods make changes local to the failed action and surrounding actions; the last two replan at either the next higher level of plan abstraction or at the top level. All of these methods make structural changes to plans in progress and are applicable to nearly all failures. These recovery methods, or ones very like them, have been used in other recovery systems. WATA is similar to the "retry" method described in [8]. RV and RA are Phoenix specific forms of SIPE's Reinstantiate [20]. SA is similar to "try a different action" in SIPE [21] and action substitution used for debugging in GORDIS [23]. The two replan methods, RP and RT, are constrained forms of the more general replanning done in many failure recovery systems.

## 2.2 Evaluation: Tailoring Failure Recovery to Phoenix

Failure recovery was added to the Phoenix planner by gradually broadening the classification of failures and systematically adding new recovery methods to address them, as shown in the recovery methods as they were added. The key to the design process was maintaining the

effect of new methods by evaluating the performance of the entire set.

The methodology was to define performance in terms of an expected cost model and use the model to direct improvements to the design. The expected cost model accumulates the cost of trying enough methods to repair the failure: this amounts to the cost of trying the first method plus the cost of trying a second method if the first method fails plus the cost of trying the third and so on until no methods remain. When no methods remain, then the cost is the cost of outright failure. This combination of costs are captured in the following equation:

$$C(S_i) = C(M_a) + (1 - P(M_a|S_i))[C(M_b) + \dots(1 - P(M_g|S_i))[C(M_z) + (1 - P(M_z|S_i))[C_F]]\dots] \quad (1)$$

where  $C(S_i)$  is the expected cost of recovery for failure  $S_i$ ;  $C(M_a)$  is the cost of employing an applicable method  $a$ ;  $C_F$  is the cost of failing to recover; and  $P(M_z|S_i)$  is the probability of method  $a$  succeeding in failure  $S_i$ . Cost is measured in seconds of simulation time required to repair a plan using the method.

The performance of failure recovery was assessed in a series of three experiments [10]. The first experiment tested the assumptions upon which the model is based and derived performance baselines for failure recovery in the Phoenix environment. The second compared two approaches to selecting recovery methods: choosing randomly and choosing based on a strategy derived from the model. The third experiment compared two sets of recovery methods: the original set and the same set with two new domain specific recovery methods added. Each of the experiments collected data on the failure recovery of both fireboss and bulldozer agents. Because the bulldozers do far less planning than the fireboss, the bulldozer results tend to be similar, but less interesting. Consequently, only the results of the fireboss are reported.

### 2.2.1 Experiment 1: Baselines for Performance

This experiment gathered baselines for the parameters in the performance model and tested the assumptions of that model. The model is based on three assumptions:

1.  $C(M_m)$  is independent of the order of execution of the recovery methods.

Having tried one recovery method which failed should not cause other methods to be more or less expensive to execute.

2.  $P(M_m|S_i)$  is independent of the order of execution of the recovery methods. If

this assumption is true, then whether a recovery method is tried after another fails should have no effect on whether the new methods succeeds.

3. The cost of each method  $C(M_m)$  is independent of the situation  $S_i$ . Because the recovery methods are designed to be domain-independent, intuition suggests that their costs may be independent of when they are used.

This experiment consisted of 116 trials, in which Phoenix fought three fires over the course of 60 simulation hours, resulting in 2462 failure situations and 5558 attempts to recover from the failures. The three fires were set at intervals of eight simulation hours. Approximately once an hour, the wind speed and direction were varied by  $\pm 3$  kph and  $\pm 90$  degrees. The experiment collected data on what failures occurred, what recovery methods were tried, the order in which recovery methods were tried, and the cost in simulation seconds of executing the recovery methods. The agents were given failure recovery methods as described in Section 2.1. Bulldozer agents were given an additional method for getting out of the way of encroaching fire. Recovery methods were selected randomly without replacement for each failure encountered (the one exception being that one failure type, *insufficient-progress*, could be repeated only by one of the two replan methods).

Statistical tests on the data (ANOVAs for assumptions one and three and chi-square tests for assumption two) showed that the assumptions held for a subset of the methods. In particular, the performance of the two replan methods was sensitive to whether the replans follow other methods (assumptions one and two) and to the failure situation in which they are applied (assumption three); the four remaining methods were insensitive to failure context and order of application. Hence, the assumptions held for the four methods that make constrained modifications local to the failed action, but not for the two methods that make more sweeping modifications.

### 2.2.2 Experiment 2: Selecting Recovery Methods

Failure recovery, as implemented for experiment 1, selected recovery methods at random without replacement to repair each failure. Given the model presented in equation 1 and the values for  $C(M_m)$  and  $P(M_m|S_i)$  gathered in experiment 1, we can guide the selection of recovery methods to minimize cost. Simon and Kadane [24] showed that, for problems of the type described by equation 1, the expected cost of executing a sequence is minimized by the strategy of trying the methods in decreasing order of

$$\frac{P(M_m|S_i)}{C(M_m)}$$

which intuitively means "select the method that is most likely to succeed with the lowest cost".

We added the selection strategy to failure recovery for Phoenix and then reran the same experiment scenario as in experiment 1. Experiment 2 included 94 trials which resulted in 2001 failure situations and 3877 attempts to recover from the failures. The costs of recovering from each failure type in this experiment were compared to the results from experiment 1. Table 3 shows the mean costs of failure recovery for each failure type and the percentage overall for

Table 3: Fireboss failures in the Baseline and Strategy experiments.

	ccp	ccv	cfp	fne	ip	ner	nrs	prj	ptr	ru
Exp. 1 Costs (Random Strategy)	1932	5710	3030	1163	3883	3395	2904	2165	1373	3254
Exp. 1 $P(S_i)$	.268	.006	.118	.002	.213	.159	.002	.080	.075	.077
Exp. 2 Costs (Selection Strategy)	1056	1618	2707	474	3977	2838	414	2518	1041	2514
Exp. 2 $P(S_i)$	.212	.007	.084	.002	.203	.163	.004	.223	.043	.058

each failure type<sup>1</sup>, for experiments 1 and 2. The mean recovery cost for the fireboss was 2943 for Experiment 1 ( $sd = 3038$ ,  $n = 1053$ ) and 2500 for Experiment 2 ( $sd = 4024$ ,  $n = 1026$ ). A z-test on the differences between the mean recovery costs for the fireboss in the two experiments yielded a significant result ( $z = -2.83$ ,  $p < .0023$ ). Because not all of the assumptions of the model held, the selection strategy is not guaranteed to be optimal, but based on these results, it appears to significantly reduce the overall cost of failure recovery.

### 2.2.3 Experiment 3: Tailoring the Method Set

The set of recovery methods described in Section 2.1 are general and intended for use in most planning environments. This experiment tested whether we could further improve performance by designing recovery methods specifically for those failures that were being handled inadequately by the current set. Two new methods were added for each of the agents. These methods were designed for failures that were both expensive and frequently occurring: *ip*, *prj* and *ner* for the fireboss. The new methods were based on existing methods and were tailored to Phoenix.

For this experiment, 84 trials of Phoenix were run using the same experiment scenario

<sup>1</sup>The large increase in the percentage of *prj* failures is due, at least in part, to the introduction of a programming bug that erroneously detected *prj* failures.

as in the previous experiments: failure recovery incorporated the selection strategy used in experiment 2 with the two new methods added. These trials resulted in 1540 failure situations and 4279 attempts to recover from the failures. As Figure 6 shows, the overall costs of failure recovery for the fireboss decreased in the three failure situations addressed by the two new methods, but the costs *increased* in all but one of the other failure situations. Because the three targeted failures occurred frequently, this resulted in a reduction of the mean cost over all failure situations (from 2500 to 2355), but the reduction was not statistically significant.

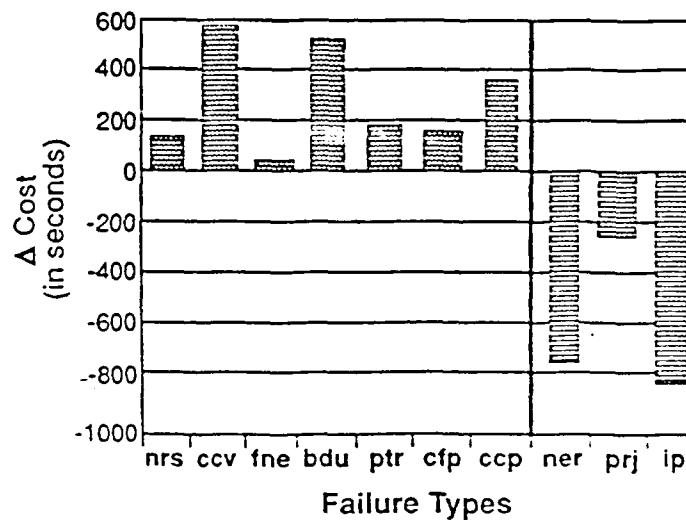


Figure 6: Cost changes from Experiment 2 to Experiment 3.

### 2.3 Summary of Failure Recovery

The methodology followed for implementing failure recovery was to construct a basic set of general recovery methods and a model of expected cost and then run a series of experiments to test assumptions about performance and expectations of how to improve performance, gradually refining failure recovery to address the requirements of the environment. Based on the results of the Baseline experiments, it appears that an untuned set of methods performs reasonably well

at least in Phoenix, and the method set can be tuned, within limits, to suit the environment better. The advantage of constructing failure recovery iteratively using a model is that the system achieves a basic level of performance quickly and at each stage, both the failure recovery implementation and our understanding of what works improved.

In terms of constructing failure recovery for some other environment and planner, the most important result from these experiments is the insensitivity of certain properties of some general methods (the local methods) to aspects of their execution context: cost is independent of position and failure situation, and probability of success for a situation is independent of position of execution. The result is important both for design and methodological considerations. As designers, we know that if such independence assumptions hold in other planners, then we can use a similar ordering strategy and predict performance for these local methods. Furthermore, given the methodology of directing evaluation using a model, we can state and test some of the assumptions of our designs.

The implication of the experiment results for designers of failure recovery is that it can be extremely difficult to control the effects of even small changes to failure recovery. For example, two methods were added to assist in recovering from three types of failures. Yet, these changes had unexpected consequences: the type and frequency of failures changed (some increased, some decreased) and the cost of recovering from failures not addressed by the new recovery methods *increased*. The incidence of failures changed because the new recovery methods were preventing some failures, causing others, and permitting some plans to get further along before they failed. Because the recovery methods make mostly syntactic changes to the plan based on the content of the plan library, we suspected that it was not the recovery methods themselves causing the failures, but rather parts of the plan library. From this intuition developed the analysis method described in the next section.

### 3 Debugging the Planner

Failure recovery is an approach for dealing with the unexpected source of failure. Typically, we prefer to avoid failures rather than patch up the plans after the failures. Some failures are difficult to avoid because the environment is capricious or the contingencies are simply too expensive. The failures that seem most amenable to avoiding (i.e., those over which we have the most control) are those caused by bugs in the planner itself: debugging the planner removes such sources of failure.

Most AI approaches to debugging planners are knowledge-intensive. Sussman's HACKER [27] detects, classifies and repairs bugs in Blocksworld plans, but it requires considerable knowledge about its domain. Hammond's CHEF [7] backchains from failure to the states that caused it, applying causal rules that describe the effects of actions and maps the information to canonical failures and repair strategies. Simmons' GORDIUS [23] debugs faulty plans by regressing desired effects through a causal dependency structure constructed during plan generation from a causal model of the domain. Kambhampati's approach [11] requires the planner to generate validation structures, explanations of correctness for the plan. His theory of plan modification compares the validation structure to the planning situation, detects inconsistencies, and uses the validation structure to guide the repair of the plan.

The approaches described so far are comprehensive: they all analyze a plan for what went wrong and repair the plan or model to avoid the failure in the future. Thus, they all assume that the analysis mechanism possesses a complete model of the domain and/or planner. Alternatively, the debugger could solicit outside help or information by asking a human user or by inferring obtained from failure recovery execution. Broverman [3] views failure recovery as an opportunity for knowledge acquisition and requests assistance from the human user of the planner to augment the system's model. Zito-Wolf and Alterman adapt plans when overly general

plans fail [30]; these adaptations repair failures and then are used to augment the plan stored in long-term memory with choice points and alternative actions. In these approaches, we may sacrifice completeness, or degree of automation, to broaden the types of failures addressed.

### 3.1 Failure Recovery Analysis: An Approach to Debugging the Plan Knowledge Base

Failure Recovery Analysis exploits information about *observed* relationships between failures and repairs to help the designer discover how the planner or failure recovery may be causing failures. Because it relies primarily on execution data, FRA requires little knowledge to identify contributors to failures and only a weak model to explain how the planner might have caused failures. Complementary to the more knowledge-intensive approaches, FRA is most appropriate when a rich domain model is not available or when the existing model might be incorrect or buggy, as when the system is under development.

FRA is a partially automated procedure that is controlled by a human designer, rather than a fully automated system. The designer participates at every step in the process, deciding where to focus attention and ultimately how to fix the planner to repair the bug. FRA trades power for generality. Because it uses a weak model, it can locate a variety of bugs and should be appropriate for many planners, but it cannot guarantee that it will find all bugs or correctly explain the failures.

In FRA, plan debugging proceeds as an iterative redesign process in which the designer tests the design, locates flaws, and modifies the planner to remove the flaws, as in Figure 1. The process continues until the designer is satisfied that enough flaws have been removed.

**Step 1: Run Planner** The designer starts by running the planner to test the current design, collecting execution traces of what failures occurred and how they were recovered. Recall that

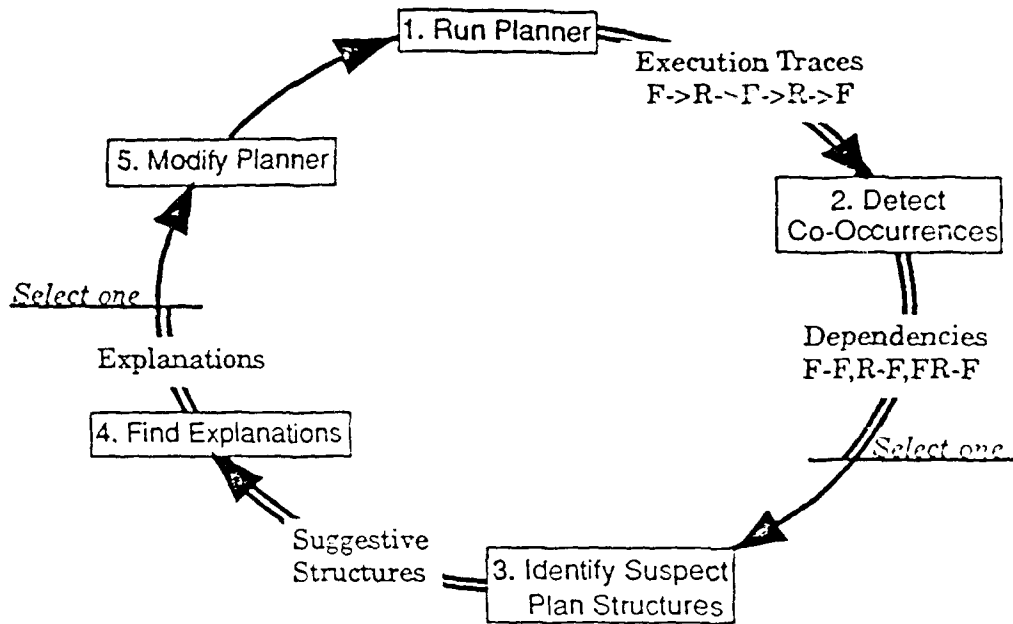


Figure 7: Failure Recovery Analysis: The Designer iteratively debugs the planner by testing the planner, locating flaws and modifying the planner to remove the flaws.

this was part of the information collected in the experiments described in Section 2.2.4. An *execution trace* is an abstracted view of the interactions between the planner and its environment. For purposes of debugging, we can view the operation of a planner in its environment as a series of plan failures followed by recovery actions taken by the planner, as in the following trace:

$$F_{1,p} \rightarrow R_{1,p} \rightarrow F_{2,p} \rightarrow R_{2,p} \rightarrow F_{3,p} \rightarrow R_{3,p} \rightarrow F_{4,p}$$

where  $F$ 's are environment states and  $R$ 's are actions. The subscripts indicate individuals from a set, so  $R_{sp}$  means recovery action of type  $sp$ . It appears from this short trace that failure  $F_{ip}$  is always preceded by the recovery action  $R_{sp}$ , but without more examples we cannot be confident of the relationship.

**Step 2: Detect Co-Occurrences** The execution traces are searched (by a program) for statistically significant co-occurrences (called *dependencies*) between recovery efforts and sub-

	$F_{ip}$	$F_{ip}^-$
$R_{sp}$	52	33
$R_{sp}^-$	240	643

Table 4: Contingency Table for  $[R_{sp}, F_{ip}]$ 

sequent failures. Dependencies tell the designer how the recovery actions influence the failures that occur and how one failure influences another.

Dependencies can be detected by statistically analyzing the execution traces. The statistical analysis is a two-step process: Combinations of failures and actions are first tested for whether they are more or less likely to be followed by each of the possible failures. Then the significant combinations are compared to remove overlapping combinations.

For example, to determine whether  $F_{ip}$  is related to  $R_{sp}$ , we compare the incidence of  $F_{ip}$  after  $R_{sp}$  to  $F_{ip}$  after any action other than  $R_{sp}$ . To test the significance of the difference between the incidences, we apply a statistical test (the G-Test which is similar to the  $\chi^2$  test) to a 2x2 contingency table of the counts of  $F_{ip}$  after  $R_{sp}$ , states other than  $F_{ip}$  after  $R_{sp}$ ,  $F_{ip}$  after actions other than  $R_{sp}$ , and environment states other than  $F_{ip}$  after actions other than  $R_{sp}$ . We collect these frequencies from many execution traces and arrange them in the contingency table in Table 4. The numbers in this table (taken from the data from Experiment 3 described in the Section 2.2) show that  $F_{ip}$  follows  $R_{sp}$  far more frequently than it follows other recovery actions. A G-test on this table yields  $G = 42.9, p < .001$ , which suggests that the contingency table in Figure 4 is extremely unlikely to have arisen by chance if  $R_{sp}$  and  $F_{ip}$  are independent. So we conclude that  $F_{ip}$  depends on  $R_{sp}$  (abbreviated  $[R_{sp}, F_{ip}]$ ).

We can test dependencies between some failure and any precursor. This analysis has three precursors: recovery actions ( $R$ ), failures ( $F$ ) or pairs of a failure and the recovery action that repaired it ( $F/R$ ). If we test for dependencies in all three types of precursors, we shall find cases

of overlap: we observe dependencies involving both a particular action itself (e.g.,  $R_a$ ) and the action in combination with some failure (e.g.,  $F_f R_a$ ). To distinguish whether the action itself or the combination best describe the relationship, we can apply a variant of the G-test to determine the contribution of each combination to the effect of the action itself. In this way, we can reduce a large set of overlapping dependencies to a smaller set of mutually exclusive dependencies.

**Step 3: Identify Suspect Plan Structures** The designer selects one of the dependencies for further attention and tries to determine how the planner's actions might have related to the observed dependency. The dependencies are mapped to actions in plans and then the plans are searched for structures that involve the actions and that are known to be susceptible to failure. These structures are called *suggestive structures* because they are suggestive of possible bugs: they are language mechanisms used to coordinate actions and can be difficult to program properly. Currently, FRA includes seven suggestive structures.

For example, to associate the dependency identified in step 2,  $[R_{sp}, F_{ip}]$ , with plan actions, we determine what actions are added by  $R_{sp}$  and what actions detect  $F_{ip}$ , as displayed in Figure 8. In this case,  $R_{sp}$  adds one of three projection actions: *multiplex-shell* ( $A_{p=sp}$ ), *tight-shell* ( $A_{p=ts}$ ), and *mode-based* ( $A_{p=mb}$ ); failure  $F_{ip}$  is detected by a monitoring action called  $A_{int}$ .

To find suggestive structures, the plan library is searched for all plans that contain some combination of the actions found in the dependency. Each such plan is checked for suggestive structures that involve the actions in the dependency. In the example, the projection calculator actions ( $A_{p=mb}$ ,  $A_{p=ts}$  and  $A_{p=sp}$ ) and the monitoring action ( $A_{int}$ ) appear together in three different indirect attack plans. All three indirect attack plans include the same suggestive structures: *sequential ordering* and *shared variable* (see Figure 8).

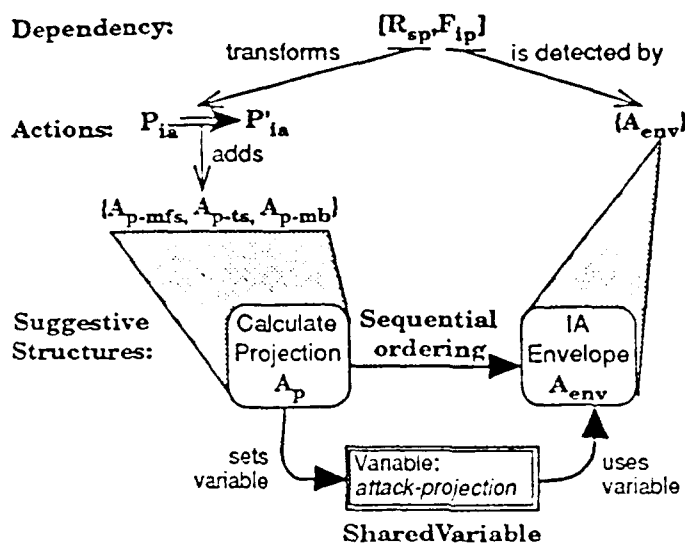


Figure 8: Mapping a dependency to two suggestive structures

*Sequential ordering* is an enabling condition: if one action is guaranteed to come before another, then the first action has the potential to influence the second. A *shared variable* requires that every action that references the variable agrees about how it is set and used. If some of the assumptions about variable's use are implicit or under-specified, the variable might be a source of failures. For example, one action might assume that the variable's units are minutes and another might assume seconds, leading to a major difference in time estimates.

Suggestive structures are similar in purpose to Thematic Organization Packages (or TOPs) in Chef [7]; however, while TOPs encompass the diagnosis of the failure, the interactions between the steps and states of the plan and the strategies for repair, suggestive structures identify only the interactions between the steps. Consequently, suggestive structures can be combined to form different explanations of failures and indicate different repairs.

**Steps 4 and 5: Find Explanations and Modify Planner** Finally, the Designer matches the suggestive structures for the dependency to a set of possible explanations and modifications.

FRA is a bit like computing a z-test in statistics with a calculator: the calculator computes the z value (for FRA, a program computes the dependency set and searches for the structures in the plan library) and you look up the significance level in a table (for FRA, the explanations and modifications).

Continuing with the example, the two suggestive structures found for the dependency  $[R_{sp}, F_{ip}]$  underlie two different explanations:

**Implicit Assumptions** Two actions make different assumptions about the value of a plan variable to the extent that the later action's requirements for successful execution are violated. This can be fixed by adding new variables to the plan description to make the assumptions explicit or changing the plans so that the incompatible actions are not used in the same plans.

**Band-aid Solutions** A recovery action may repair the immediate failure, but that failure may be symptomatic of a deeper problem, which leads to subsequent failures. This can be fixed by limiting how the recovery action is used or substituting a new recovery action.

The shared variable can cause a failure if the substituted projection calculation action sets the variable differently than was expected by the envelope action: the projection may not be specified well enough to be properly monitored or may violate monitoring assumptions about acceptable progress. Alternatively, the recovery action  $R_{sp}$  could lead to  $F_{ip}$  if the recovery action is repairing only a symptom of a deeper failure: the fire may be raging out of control or the available resources may really be inadequate for the task.

The explanations amount to sketches of what might have gone wrong. They do not precisely determine the cause, but rather attempt to provide enough evidence of flaws in the recovery actions or the planner to help the designer decide how to fix the bug. The modification is left to the designer. In effect, the selected explanation and modification constitute a hypothesis

about what caused the failure. Repeating the cycle tests the hypothesis: in the next cycle, the designer can search for more flaws to fix and can determine whether the modification achieved the desired effect: the observed dependency disappeared and the incidence of failure changed for the better.

### 3.2 Evaluating Failure Recovery Analysis

We evaluated FRA in two ways: by demonstrating its utility in finding and fixing a bug in the Phoenix plan library and by testing the effect of sample size (size of the execution traces) on the results. The demonstration shows that FRA is useful, and the modifications based on it are effective at reducing the incidence of a particular failure. Sample size is important because dependency detection is a statistical technique and because these traces can be expensive to collect.

#### 3.2.1 Completing a Cycle of FRA

Section 3.1 described FRA in the context of an example: explaining why failure  $F_1$  tends to follow recovery action  $R_{11}$ . Modifications to the Phoenix planner have been implemented based on the example analysis and tested by starting another cycle of FRA: the new results show that the modifications improve the planner's performance.

The modification was one of two suggested by FRA. The first, which fixes a *band-aid solution*, requires limiting the application of the suspect recovery action. In this case, the recovery action had been added to improve recovery performance in two expensive failures; removing it would set performance back to previous levels. The other modification was intended to fix *incorrect assumptions*. This modification has two parts: first, check how the projection calculation actions set and the monitoring action uses the variable *attack-projection*, and second, reconcile any differing assumptions of the three actions that set the variable so that the monitoring action

can use the assumptions.

The second modification was implemented: checking the code for the projection actions and the monitoring action showed that the values set by the three projection actions varied widely. The monitoring action uses summaries of the resources' capabilities, set by the projection actions, to construct expectations of progress for the plan. The three projection actions differed in what capabilities were included in those summaries (e.g., rate of building fireline, rate of travel to the fire, startup times for new instructions, and refueling overhead). Because the monitoring action assumed that the summaries reflected *only* the rate of building fireline, the conditions for signaling failures effectively varied among the different projection actions. To accommodate these differences, the projection actions were restructured to set separate variables for each of the capabilities: the monitoring action then combines the separate variables to define expected progress. In addition, a few minor bugs were fixed in the calculation of the summaries.

The modified planner was tested in 87 trials of the same experiment setup used for the earlier three experiments and analyzed for dependencies. If the recommended modification was appropriate, the observed  $[R_{tr}, F_{tr}]$  dependency should have disappeared, as well as other dependencies involving projection actions and the monitoring action. In fact, all four of the dependencies involving projection actions and the envelope action previously observed —  $[R_{tr}, F_{tr}]$ ,  $[F_{tr}, R_{tr}, F_{tr}]$ ,  $[F_{tr}, R_{tr}, F_{tr}]$ , and  $[F_{tr}, R_{tr}, F_{tr}]$  — were missing from the modified planner's execution traces. Additionally, fixing the calculation bugs and restructuring the actions led to a lower incidence of a general failure to calculate projections ( $F_{proj}$ ;  $F_{proj}$ ) accounted for 29.8% of the failures in the previous experiment and only .3% in this experiment. By repairing a hypothesized cause of failure, one would also expect the overall rate of failures to decline. The data showed a decrease in the mean failures per hour from .44 in the previous experiment to .33 in this experiment.

As someone who had worked on the Phoenix planner for some time, this example was

interesting for two reasons: First, there is a long time delay between the execution of the recovery action and the failure detection, which means that we had not thought to check the much earlier event as a source of failures or bugs. Second, the projection actions (the actions added by  $R_{sp}$ ) involve extremely complex code, which is difficult to debug. Consequently, the information provided by FRA was useful in tracking down bugs in the projection code.

### 3.2.2 Sensitivity of Dependency Detection to the Size of Execution Traces

Execution traces are often expensive to collect. Consequently, much of the effort required to execute dependency detection is expended collecting execution traces. We expect that the results of dependency detection will vary based on how many execution traces we collect: the total number of patterns (i.e., possible combinations of different types of precursors and failures) and the ratios of the patterns (i.e., the ratio of the counts in the first column to the counts in the second column) in a contingency table influences the results of the G-test. To determine how the size and number of execution traces collected influences the results of the G-test, we need to answer two questions: How does the value of G change as the number of patterns in the execution traces increases? How does the value of G change as the precursor to failure co-occurrence (i.e., the ratio of the upper right to upper left cells in the contingency table) varies from the rest of the execution traces (i.e., the ratio of the lower left to the lower right cells in the contingency table)? The first question addresses the sensitivity of the test to the size of the execution traces; the second addresses the sensitivity to noise: how much of a difference is required to detect a dependency?

**G-Test Sensitivity to Execution Trace Size** We selected the G-test over the more common Chi-square test because the G-test is additive. Additivity means that G values for subsets of the sample can be added together to get a G value for the superset. If the ratios remain the

same but the total number of counts in the contingency table double, then the G value for the contingency table doubles as well. Consequently, given execution traces with few patterns, the G-test can find strong dependencies, but given more patterns, it will also find rare dependencies. If a user of FRA is interested in detecting any dependencies, then a few execution traces will be adequate to do so; if the user wishes to find rare or obscure dependencies, then it will be necessary to gather more execution traces. The level of effort expended in gathering execution traces depends on what kinds of dependencies one wishes to find.

**G-Test Sensitivity to Noise** We know that the value of G increases linearly with increases in the number of patterns in the execution traces, but only if the ratios in the contingency table remain the same, as the number of patterns increases. In trying to decide how many execution traces to gather, we also need to know whether the results will be vulnerable to noise, which is more apparent with few patterns. We can evaluate empirically whether, in practice, getting slightly more or fewer execution traces would have significantly changed which dependencies were detected in execution traces gathered from Phoenix.

We tested sensitivity to noise by "tweaking" the frequency counts found in the data to see how many of the dependencies would not have been detected if the counts in row one in the contingency table varied by a small amount. For four sets of execution traces (three from the three experiments plus the set from the fourth experiment described in the last section), tweaking the values resulted in a loss of about 35% of the dependencies found previously. In other words, for the data from Phoenix, dependency detection is sensitive to small differences in the content of the execution traces. Most of the dependencies that were vulnerable to the tweaking were based on execution traces that included few instances of the precursor failure pattern. 52% of the dependencies that disappeared were based on contingency tables in which one of the counts in the first row was less than five.

The implication of the sensitivity of dependency detection to noise in the execution traces is that rare patterns are especially sensitive to noise and so should be viewed skeptically. One must interpret the results of dependency detection with care: if "sensitive" dependencies are discarded, then rare events may remain undetected; at the same time, one does not wish to chase chimeras. Interpreting dependencies requires weighing false positives against misses. If we are trying to identify dependencies between precursors that occur rarely or failures that occur rarely, then additional effort should be expended to get enough execution traces to ensure that the dependency is not due to noise.

### 3.3 Summary

The purpose of Failure Recovery Analysis is to identify cases in which plans may influence, exacerbate or cause failure. There are two reasons why analyzing failure recovery is a good way to determine why the planner fails. First, failure recovery influences which failures occur. Minor changes in the design of failure recovery produce significant changes in the number and types of failures. Second, failure recovery uses plans in ways not explicitly foreseen by its designers, but not forbidden or prevented by them either. Failure recovery repairs plans by adding or replacing portions of them. As a result, the plan may include plan fragments that are juxtaposed in orders and context not envisioned by the designers.

The current results are preliminary. FRA has been tested on a small number of examples from a single planner. Because it is based on few assumptions about the planner and its environment, it should easily generalize to planners for which execution traces are available.

The most interesting feature of FRA is the integration of knowledge-based and statistical reasoning to overcome some of the limitations of current computational systems. Statistics provides useful summaries of performance and data, but requires interpretation. Knowledge-based techniques in the later steps of FRA provide partial interpretation of statistics.

results.

## **4 Future Work**

The FRA procedure promises to improve planner reliability and to expedite the development of plan knowledge bases for new environments by assisting designers in debugging knowledge bases. However, at its present stage of development, FRA is limited in several ways: the procedure is only partially automated and is implemented as a loosely organized set of Lisp functions, execution traces contain only failures and recovery actions, dependencies include only temporally adjacent precursors and failures, and the procedure has been tested only on the Phoenix planner.

Future research will address these limitations by "closing the loop" of gathering and analyzing execution data and by generalizing FRA to a broader range of bugs and to another planner. Closing the loop refers integrating all of the tools necessary to support complete testing, analysis and repair of a planner during its development process. The designer will still direct the process, but will do so by selecting from sets of pre-defined experiment scenarios and scripts for performing stereotypical analysis of the execution data. Generalizing FRA refers to expanding the set of bugs that can be identified and applying the procedure to another planner. The set of suggestive structures and explanations needs to be enhanced, especially when FRA is applied to another planner and environment. The range and nature of dependencies needs to be further explored as well. For example, if dependencies reflect the interaction between the planner and its environment, will similar environments and planners result in similar patterns of dependencies? Dependencies may prove to be a metric of similarity between different planners.

## 5 Conclusion

Certain software systems, so called *ambitious systems* [5], are prone to failure. These include systems being developed for novel or unfamiliar tasks, systems in unpredictable environments, or systems with organizational complexity. Failure is a consequence of complexity in the environment or the software and the fact that our facility in constructing complex systems has surpassed our ability to understand their behavior. Consequently, the software most likely to fail is that which is also hardest to understand and to debug.

The goal of the described research is to reduce the impact and likelihood of failures that result from a lack of understanding about how an AI planner will perform. Failure recovery provides a safety net for catching failures that cannot be avoided easily: the incremental methodology expedites building failure recovery suited to a particular planner and its environment. Failure Recovery Analysis helps programmers to debug planners under development because it requires only a weak model of how they perform and relies on statistical analyses of the execution of the planner. Together, these approaches have been demonstrated to improve the reliability and the performance of the Phoenix planner. Given how few assumptions are made about the planner and its environment, FRA promises to do so for other planners as well.

## References

- [1] Jose A. Ambros-Ingerson and Sam Steel. Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 83-88. Minneapolis, Minnesota, 1988. American Association for Artificial Intelligence.
- [2] Rodney A. Brooks. Symbolic error analysis and robot planning. *International Journal of Robotics Research*, 1(4):29-68, Winter 1982.
- [3] Carol A. Broverman. *Constructive Interpretation of Human-Generated Exceptions During Plan Execution*. PhD thesis, COINS Dept, University of Massachusetts, Amherst, MA, February 1991.
- [4] Paul R. Cohen, Michael Greenberg, David M. Hart, and Adele E. Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3), Fall 1989.
- [5] Fernando J. Corbato. On building systems that will fail. *Communications of the ACM*, 34(9):72-81, September 1991.
- [6] Maria Gini. Automatic error detection and recovery. Computer Science Dept, 88-48, University of Minnesota, Minneapolis, MN, June 1988.
- [7] Kristian John Hammond. *Case-Based Planning: An Integrated Theory of Planning, Learning and Memory*. PhD thesis, Dept. of Computer Science, Yale University, New Haven, CT, October 1986.
- [8] Steve Hanks and R. James Firby. Issues and architectures for planning and execution. In Katia P. Sycara, editor, *Proceedings of the Workshop on Innovative Approaches to*

- Planning, Scheduling and Control*, pages 71-76. Palo Alto, Ca., November 1990. Morgan Kaufmann Publishers, Inc.
- [9] Adele E. Howe. *Accepting the Inevitable: The Role of Failure Recovery in the Design of Planners*. PhD thesis, University of Massachusetts, Department of Computer Science, Amherst, MA, February 1993.
- [10] Adele E. Howe and Paul R. Cohen. Failure recovery: A model and experiments. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 801-808. Anaheim, CA, July 1991.
- [11] Subbarao Kambhampati and James A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence Journal*, 55(2-3), 1992.
- [12] Victor R. Lesser. A retrospective view of FA/C distributed problem solving. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1347-1362, November/December 1991.
- [13] Nancy G. Leveson. Software safety: Why, what, and how. *Computing Surveys*, 18(2):127-164, June 1986.
- [14] D.H. Long, S. R. Vijayakumar, and S. I. Venkataratnam. A representation for error detection and recovery in robot plans. In *Proceedings of SPIE Symposium on Intelligent Control and Adaptive Systems*, pages 14-25. Philadelphia, November 1989.
- [15] David R. Miller. Execution monitoring for a mobile robot system. In *Proceedings of SPIE Symposium on Intelligent Control and Adaptive Systems*, pages 36-43. Philadelphia, PA, November 1989.
- [16] Steve Stanton, Mark D. Johnston, Andrew B. Phillips, and Philip Land. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In

- Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 17-24. Anaheim, CA, 1991. American Association for Artificial Intelligence.
- [17] Leora Morgenstern. Replanning. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, pages 5-1 - 5-10. Austin, TX, December 1987.
- [18] N. Hari Narayanan and N. Viswanadham. A methodology for knowledge acquisition and reasoning in failure analysis of systems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-17(2):274-288, March/April 1987.
- [19] S.Y. Nof, O.Z. Maimon, and R. G. Wilhelm. Experiments for planning error-recovery program in robotic work. In *Proceedings of the 1987 ASME International Computers in Engineering Conference*, pages 253-262. NY,NY, August 1987.
- [20] Christopher Owens. Representing abstract plan failures. In *Proceedings of the Twelfth Cognitive Science Conference*, pages 277-284. Boston, MA, 1990. Cognitive Science Society.
- [21] Harry J. Porta. Dynamic replanning. In *Proceedings of ROBEXS 86: Second Annual Workshop on Robotics and Expert Systems*, pages 109-115, June 1986.
- [22] Reid Simmons. Monitoring and error recovery for autonomous walking. In *Proc. III International Workshop on Intelligent Robots and Systems*, pages 1407-1412, July 1992.
- [23] Reid G. Simmons. A theory of debugging plans and interpretations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 94-99, Minneapolis, Minnesota, 1988. American Association for Artificial Intelligence.
- [24] Herbert A. Simon and Joseph B. Kadane. Optimal problem-solving search: All or none solutions. *Artificial Intelligence Journal*, 6:235-247, 1975.

- [25] Stephen F. Smith, Peng Si Ow, Nicola Muscettola, Jean-Yves Potvin, and Dirk C. Matthys. OPIS: an integrated framework for generating and revising factory schedules. In Katia P. Sycara, editor, *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 497-507. Morgan Kaufmann Publishers, Inc. November 1990.
- [26] Sankaran Srinivas. *Error Recovery in Robot Systems*. PhD thesis, California Institute of Technology, Pasadena, CA, 1977.
- [27] Gerald A. Sussman. A computational model of skill acquisition. Technical Report Memo no. AI-TR-297, MIT AI Lab, 1973.
- [28] Katia Sycara. Using case-based reasoning for plan adaptation and repair. In *Proceedings of a Workshop on Case-Based Reasoning*, pages 425-434. Morgan Kaufmann Publishers, Inc., 1988.
- [29] David E. Winkler. Recovering from execution errors in SIPE. Technical Report 140, Artificial Intelligence Center, Computer Science and Technology Center, SRI International, 1984.
- [30] Robert J. Frost and Richard Alterman. Ad-hoc, fall-safe plan learning. In *Proceedings of the 1990 Cognitive Science Conference*, pages 908-913, Boston, MA, July 24-28 1990.

#### 4. An Empirical Method for Constructing Slack time Envelopes

From *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society* (1992).

### Early Warnings of Plan Failure, False Positives and Envelopes: Experiments and a Model

Paul R. Cohen, Robert St. Amant, David M. Hart

#### Abstract

We analyze a tradeoff between early warnings of plan failures and false positives. In general, a decision rule that provides earlier warnings will also produce more false positives. Slack time envelopes are decision rules that warn of plan failures in our Phoenix system. Until now, they have been constructed according to ad hoc criteria. In this paper we show that good performance under different criteria can be achieved by slack time envelopes throughout the course of a plan, even though envelopes are very simple decision rules. We also develop a probabilistic model of plan progress, from which we derive an algorithm for constructing slack time envelopes that achieve desired tradeoffs between early warnings and false positives.

#### 1 Introduction

Underlying the judgment that a plan will not succeed is a fundamental tradeoff between the cost of an incorrect decision and the cost of evidence that might improve the decision. For concreteness, let's say a plan succeeds if a vehicle arrives at its destination by a deadline, and fails otherwise. At any point in a plan we can correctly or incorrectly predict that the plan will succeed or fail. If we predict early in the plan that it will fail, and it eventually fails, then we have a *hit*, but if the plan eventually succeeds we have a *false positive*. False positives might be expensive if they lead to replanning. In general, the false positive rate decreases over time (e.g., very few predictions made immediately before the deadline will be false positives) but the reduction in false positives must be balanced against the cost of waiting to detect failures. Ideally, we want to accurately predict failures as early as possible; in practice, we can have accuracy or early warnings but not both.

The false positive rate for a decision rule that at time  $t$  predicts failure will generally decrease as  $t$  increases. We analyze this tradeoff in several ways. First, we describe a very simple decision rule, called a *slack time envelope*, that we have used for years in the Phoenix planner (Sections 2 and 3). Then, using empirical data from Phoenix, we evaluate the false positive rate for envelopes and show that envelopes can maintain good performance throughout a plan (Section

4). An infinite number of slack time envelopes can be constructed for any plan, and the analysis in Section 4 depends on "good" envelopes constructed by hand. To be generally useful, envelopes should be constructed automatically. This requires a formal model of the tradeoff between when a failure is predicted (earlier is better) and the false positive rate of the prediction (Section 5). Finally we show how the conditional probability of a plan failure given the state of the plan can be used to construct "warning" envelopes.

#### 2 Slack Time Envelopes

Imagine a plan that requires a vehicle to drive 10 km in 10 minutes. Figure 1 shows progress for three possible paths that the vehicle might follow, labeled A, B and C. Case A is successful: the vehicle makes rapid progress until time 3, then slows down from time 3 to time 4, then makes rapid progress until time 8, when it completes the plan. Case B is unsuccessful: progress is slow until time 4, and slower after that; and the required distance is not covered by the deadline. The solid, heavy line is a slack time envelope for this problem. Our Phoenix planner (Cohen et al., 1989; Hart, Anderson, & Cohen, 1990) constructs such an envelope for every plan and checks at each time interval to see whether the progress of a plan is within the envelope. Case A remains within the envelope until completion; case B violates the envelope at time 6.

The solid, heavy line is a slack time envelope for this problem. Our Phoenix planner (Cohen et al., 1989; Hart, Anderson, & Cohen, 1990) constructs such an envelope for every plan and checks at each time interval to see whether the progress of a plan is within the envelope. Case A remains within the envelope until completion; case B violates the envelope at time 6.

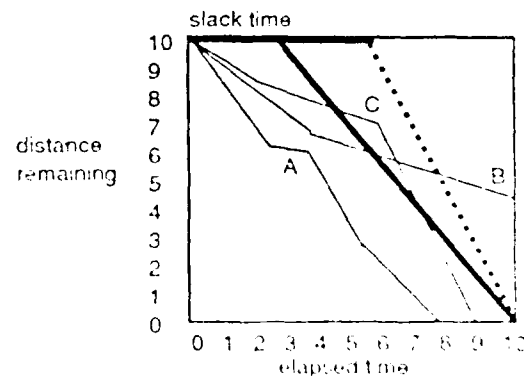


Figure 1. Illustration of envelopes

When an envelope violation occurs, the Phoenix planner modifies or completely replaces its plan. It should not wait until the deadline has expired to begin, but should start replanning as soon as it is reasonably sure that the plan will fail. Clearly, envelopes can provide early warning of plan failure; for example, in case B, the envelope warned at time 6 that the plan would fail. The problem is that progress might pick up after an envelope violation, as shown in case C. At time 5 the envelope is violated, but by time 8, the plan is back within the envelope. If in this case the Phoenix planner abandoned its plan at time 5, it would have incurred needless replanning costs. Case C is a false positive as we defined it earlier: a plan predicted to fail that actually will succeed. Note that a different envelope, shown by the heavy dotted line, will avoid this problem. Unfortunately, it doesn't detect the true failure of case B until time 8, two minutes after the previous envelope. This illustrates the tradeoff between early warnings and false positives. (This and other concepts in the paper derive from signal detection theory, e.g., (McNicol, 1972; Coombs, Dawes, & Tversky, 1970).)

Slack time envelopes get their name from the period of no progress that they permit at the beginning of a plan. The Phoenix planner adds slack time to envelopes so that plans will have an opportunity to progress before they are abandoned for lack of progress. Until recently, this was all the justification for envelopes we could offer. In the following sections, however, we show why the simple linear form of envelopes achieves high performance, and how to select a value of slack time.

### 3 The Data Set

One way to evaluate slack time envelopes is to generate hundreds of plans, monitor their execution at regular intervals, and, at each interval, use an envelope to predict success or failure. We generated 1139 travel plans, or *paths*, for vehicles in our Phoenix simulation. Phoenix is based on a machine-readable map of Yellowstone National Park that includes roads, obstacles, a variety of elevations and ground covers, and other terrain features. The Phoenix planner fights simulated forest fires in this environment by surrounding the fires with fireline built by bulldozers. Envelopes in Phoenix monitor fire spread rate, fireline digging, and progress in different bulldozer tasks. The focus of this paper, however, is a simpler problem: getting from one point on the map to another by a deadline. To generate our data set, we repeatedly selected pairs of points 70 km apart as the crow flies, and asked the Phoenix planner to construct a path between each pair. Then we simulated the traversal of each path, monitoring it every 1000 simulated seconds. At each monitoring step we estimated the distance remaining to the destination. Because of obstacles, terrain, and so on, the distribution of remaining distances at a given monitoring interval

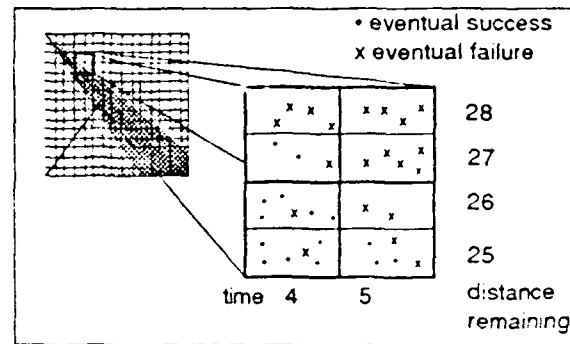


Figure 2. How we generated distributions of DR for successes and failures at each time interval.

was considerable (including many greater than 70 km). For example, after 5000 seconds, the mean remaining distance was about 54 km with a range of 13.6 to 79.1 km. We generated, executed and monitored 1139 paths in this manner.

#### 3.1 Distributions of Eventual Successes and Failures Before the Deadline

We chose a deadline of 15,000 seconds to divide the paths into two groups: paths that reached their goals by the deadline were called *successes*, and those that did not were called *failures*. Of 1139 paths, 654 succeeded and 485 failed. We looked at each path 15 times, once every 1000 seconds, and recorded an estimate of the number of "distance units" remaining to the goal. For a variety of reasons, a distance unit is 2 km, so the distance remaining to the goal, abbreviated DR, is 35 at the beginning of the plan and zero for successful paths at the end of the plan. Henceforth, we use "time  $x$ " as shorthand for "x thousand seconds elapsed." For example, in Figure 2, at time 4, all the paths with  $DR = 28$  are failures; at time 5, all the paths with  $26 \leq DR \leq 28$  are failures; but at time 5,  $DR = 25$ , three paths are successes and two are failures.

We plotted frequency polygons for DR for successes and failures at each of the 15 time intervals. Figure 3 shows the distribution of successes and failures at time 5. Note that most failures still have a long way to travel at time 5: the bulk of the distribution lies to the right of  $DR = 30$  (the mean DR for failures at time 3 is 33). The distribution of successes, however, is made up of paths with relatively short remaining distances to the goal (mean  $DR = 17$ ).

#### 3.2 Empirical Hit Rates and False Positive Rates for DR Thresholds

Let's predict that a path will fail to reach its goal by its deadline if, at time 5, the remaining distance to the goal is 30 or more, that is, the *threshold*  $DR \geq 30$ . The dark shaded area in Figure 3 represents false positive

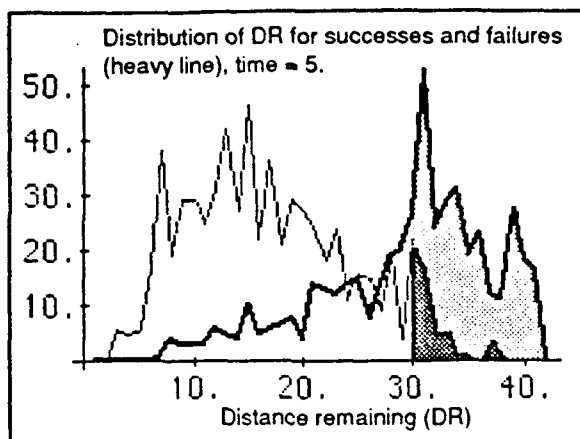


Figure 3. Frequency polygons for DR at time 5.

errors, paths we predict will fail but that eventually succeed. Of the 654 paths that eventually succeed, 37 lie in the dark shaded area: the probability of a false positive is therefore  $37/654 = 0.056$ . The light shaded area represents hits, paths that are predicted to fail and that actually do fail. The probability of a hit is  $261/485 = 0.538$ . The ratio of the probabilities is 9.60.

At time 5, the threshold  $DR \geq 30$  seems pretty good because the ratio of hit probability to false positive probability is high, but we cannot say it is the *best* threshold unless we know the relative values of hits and false positives.

This is just part of the analysis of our data set. In particular, we haven't shown our analyses of success and failure distributions at other times; nor hit and false positive probabilities for different DR thresholds at different times; nor success and failure distributions for stricter or more lenient deadlines. We can summarize these analyses as follows: At later time intervals, the success distribution is increasingly right skewed, with most of its mass around low values of DR. At intermediate time intervals (e.g., time = 7) the failure distribution is roughly uniform. Later, it is right skewed like the success distribution, but with more mass in its tail than the success distribution. Shifting the DR threshold to the right decreases both hits and false positives, though false positives decrease faster (as in Fig. 5, only more so at later time intervals). These patterns hold for stricter and more lenient deadlines; the main effect of a stricter deadline is to reduce the number of successes. The following evaluations of envelopes are based on the 15,000 second deadline illustrated above because it produces a nearly even split between successes (654, total) and failures (485, total).

#### 4. Evaluation of Slack Time Envelopes

Slack time envelopes are decision rules for predicting whether paths will succeed or fail. As illustrated in Figure 1, if the path is within the boundary of an envelope

at a particular time, then we predict success, otherwise we predict failure. Each point on an envelope boundary specifies a DR threshold for a particular time, and so has an associated hit rate and false positive rate. In this section we use slack time envelopes to predict whether paths in our data set, discussed above, will succeed or fail. We evaluate the predictive performance of slack time envelopes according to this criterion: An envelope should provide performance approaching optimal throughout a plan.

This depends of course on our definition of optimal. Consider a decision rule based on distance remaining (dr) to the goal at time  $t$ :

$$\text{If } l(dr, t) > \beta(t), \text{ then predict plan failure}$$

$$\text{where } l(dr, t) = \frac{\Pr(dr | \text{plan fails}, t)}{\Pr(dr | \text{plan succeeds}, t)}$$

Intuitively, we have an observation of  $dr$  at time  $t$ , and we must decide whether this observation has been produced by an eventual success or failure. We base our decision on whether the likelihood is greater than the threshold  $\beta(t)$ . A basic result from signal detection theory is that the utility of this decision is maximized if

$$\beta(t) = \frac{\Pr(\text{plan succeeds}, t)}{\Pr(\text{plan fails}, t)} (P_{\text{payoff}}(t))$$

where

$$\text{Payoff}(t) = \frac{\text{Val}(\text{correct rej}, t) + \text{Cost}(\text{false pos}, t)}{\text{Val}(\text{hit}, t) + \text{Cost}(\text{miss}, t)}$$

In the simplest case,  $\beta(t)$  is constant over the course of a plan. A more realistic assessment requires analysis of the terms in  $\beta(t)$ . The first term, the prior probabilities, decreases with time, as plans begin to succeed. The second term  $\text{Payoff}(t)$  determines the relative importance of hits, false positives, correct rejections, and misses. The value of correctly predicting a plan failure decreases over time: early warnings are worth more. At the same time the cost of a false positive increases over time; if we are going to unnecessarily abandon a plan, it is better to do so early in the plan than later when we have invested a lot of time in the plan. It is more difficult to assess the value of a correct rejection and the cost of a miss, but if we assume they are constant relative to the other parameters, then the value of the second term in  $\text{Payoff}(t)$  increases over time. We consider the cases in which  $\text{Payoff}(t)$  is constant and also in which  $\text{Payoff}(t)$  increases linearly with time.

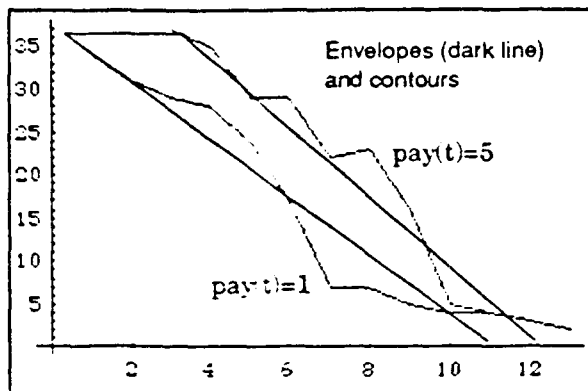


Figure 4. Hand-constructed slack time envelopes superimposed on constant  $\text{Payoff}(t)$  contours.

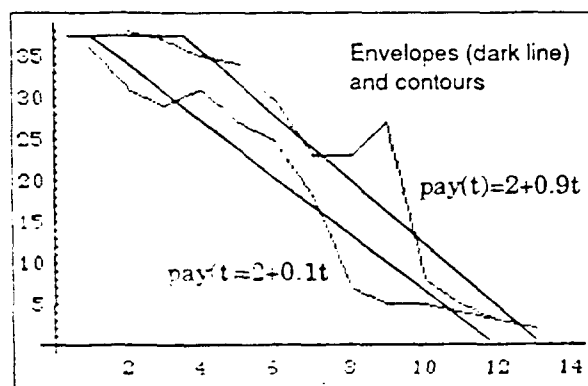


Figure 5. Slack time envelopes on linear  $\text{Payoff}(t)$  contours.

#### 4.1. Comparing Slack Time Envelopes with Empirical Utility Contours

Because envelopes are just straight lines, it is unclear whether they can satisfy the optimal performance criterion. In particular, for constant or linear payoff functions, the DR threshold required to maintain a constant ratio of hit probability to false positive probability might not change linearly over time. To find out, we calculated utility contours from the empirical data for different  $\text{Payoff}(t)$ , as shown in Figures 4 and 5. A contour represents a fixed  $\text{Payoff}(t)$  function: each point on a contour is the DR threshold (y axis) that is required at a particular time (x axis) to ensure that the utility of the decision is maximized. In Figure 4 we let  $\text{Payoff}(t)$  be constant at 1 and 5, and in Figure 5 we let  $\text{Payoff}(t)$  vary as a function of time.

An important characteristic of these contours is that they require high DR thresholds for the first few time intervals, but then gradually smaller thresholds for later time intervals. Utility contours are roughly linear, which suggests that a slack-time envelope, fit to one of these contours, could provide performance approach-

ing optimal, given our payoff function. For our data set, at least, Figures 4 and 5 tells us that an envelope can be constructed to satisfy our performance criterion. We will formalize this result in the next section.

### 5. Constructing Slack Time Envelopes: How Much Slack?

Our focus now turns to the task of constructing slack time envelopes. We assume that the end points of the envelope are the distance to the goal and the deadline, so the only parameter is how much slack time to allow. Next we present a model that predicts utility for different values of slack time. The model also predicts the *early warning premium* for values of slack time. Early warning premiums accrue when, by constructing a tight envelope with little slack time, we detect failures earlier than we would with a looser envelope. Empirically, early warning premiums come at the expense of false positives. We assess a cost for each hit proportional to the time interval in which it is detected; this places a premium on early hits. We assess a constant cost for each false positive. This is described further in the following sections.

#### 5.1. A Probabilistic Model of Progress

If we know the distributions of distance remaining (DR) for successes at each time interval (e.g., those in Figure 3) then we can predict the false positive rate for a given DR threshold. A simple model of the distribution of DR begins with the assumption that in each time interval a vehicle can progress at its maximum rate  $c$  with probability  $p$ , or makes no progress at all with probability  $q = 1 - p$ . Then the distribution of progress is binomial, as shown in Table 1: the probability of having made  $r$  units progress by time  $n$  is just the binomial probability  $\binom{n}{r} p^r q^{(n-r)}$ .

For example, the probability of one unit progress by time 4 is  $4pq^3$  because there are four ways to achieve this result, each with probability  $pq^3$ : we could make no progress until time 3 (with probability  $q^3$ ) and then progress at the maximum rate for one time unit (total probability,  $pq^3$ ). Or we could make one

	Time				
Progress	1	2	3	4	5
0c	$q$	$q^2$	$q^3$	$q^4$	$q^5$
1c	$p$	$pq$	$pq^2$	$pq^3$	$pq^4$
2c		$p^2$	$2pq$	$p^2q$	$2pq^2$
3c			$p^2$	$4pq$	$3pq^2$
4c				$p^3$	$3p^2q$
5c					$p^4$

Table 1. Progress in each time interval follows a binomial distribution.

unit of progress by time 3 (with probability  $3pq^2$ ) and then make no progress for the remaining time unit (total probability  $3pq^3$ ). The sum of these options is  $4pq^3$ .

The expected progress after  $N$  time units is  $cNp$  and the variance is  $cNpq$ . If  $p = q = .5$  then the distributions of progress in each time interval are symmetric. Otherwise the mass of the distribution at time  $N$  tends toward  $cN$  (if  $p > q$ ) or zero (if  $q > p$ ). Important characteristics of this model are that progress is linear and variance changes linearly with time.

## 5.2 Utility Contours Using the Model

This model explains the shape of utility contours and slack time envelopes, and it predicts the probability of false positives for a given envelope. Let us elaborate the model a little: Our goal is to travel some distance  $D_g$  by a deadline time  $T_g$ . At any time  $t$ , we can assess the progress that has been made,  $D(t)$ , and the progress that remains to be made,  $DR(t)$ ; and the time remaining,  $TR = T_g - t$ . A success is defined as  $D(t) \geq D_g$  and  $t \leq T_g$ . The conditional probability of a success given  $DR(t)$ ,  $D_g$  and  $T_g$  is:

$$\Pr(\text{success} | DR(t)) = \sum_{r=DR(t)}^{TR} \binom{TR}{r} p^r q^{(TR-r)}$$

A similar equation holds for the conditional probability of a failure. If  $\text{Payoff}(t)$  is for example constant, this means means that the ratio of these conditional probabilities must be constant as well. Now imagine that we have  $DR(t)$  distance remaining at time  $t$  and we extrapolate forward  $TR$  time units to the deadline. At this point we have a binomial distribution with  $N = TR$ , divided into a portion below the  $DR=0$  line (the successes, those cases that have arrived by the deadline) and a portion above the line (the failures.) The ratio of the areas of the two portions gives us the ratio of the conditional probabilities. If we want to find at each time the distance for which this ratio is constant, we plot a constant z-score for distributions with  $N$  ranging from  $T_g$  to 0.

Figure 6 shows contours for constant  $\text{Payoff}(t)$ . Contours for comparable linear  $\text{Payoff}(t)$  are very similar, with identical slack times, but more pronounced curve. To generate the figure we assumed  $D_g = 25$ ,  $T_g = 50$ ,  $p = .5$ , and  $c = 1$ , and applied the above analysis to get conditional probabilities of success and failure for every value of  $t$ .

Imagine that a vehicle has made 10 units of progress at time 25, that is,  $DR(25) = 15$ , illustrated by the large dot near the center of Figure 6. Because this dot lies on the contour labelled  $\text{Payoff}(t) = 5$ , we know that  $\Pr(\text{failure} | DR(25) = 15) / \Pr(\text{success} | DR(25) = 15) = 5$ . If the vehicle makes no progress for another

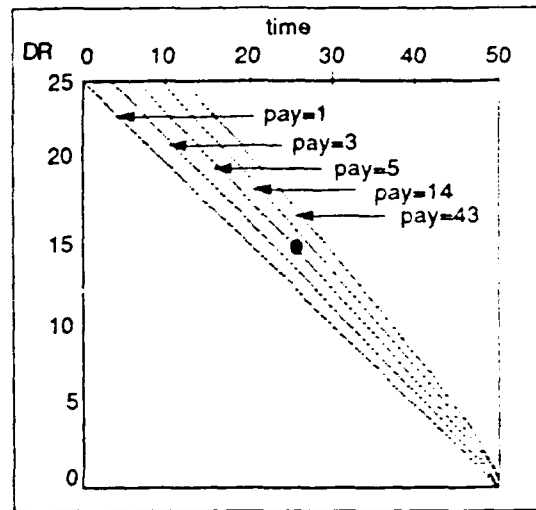


Figure 6. Contours of constant payoff from each point in the space.

five time units, then the dot would lie to the right of the contour labeled  $\text{Payoff}(t) = 43$ , so the probability ratio is much higher.

These contours vary as  $\sqrt{t}$ . At the scale on which we monitor, linear envelopes provide a good approximation of the contours, as long as the envelope boundaries have the right slope, that is, if they are constructed with the right amount of slack time. Note, too, that Figure 6 justifies the use of slack time in envelopes: The contours associated with high payoffs (and thus high ratio of hit probability to false positive probability) allow a period of no progress at the beginning of the plan.

## 5.3 Setting Slack Time

A slack time envelope is just a pair of lines, one representing the period in which no progress is required—the slack time—and another connecting the end of the first to the deadline, as shown in Figures 1 and 6. Slack time is the only parameter in slack time envelopes, but we must still show how to set it.

We desire a balance of false positives against early warning premiums. We have not yet derived from our binomial model a closed-form expression for the expected number of false positives and early warnings, but we have an algorithm that produces these expectations for a given value of slack time, if we assume that  $D_g = .5 T_g$ :

For each possible value of  $DR$ ,  $dr_i$ :

- calculate  $t_e$ , the time at which the envelope boundary will be crossed, given  $dr_i$ , for example, in Figure 1, when  $dr_i = 5$  and  $t_e = 8$ , the solid envelope boundary is crossed, so for  $dr_i = 5$ ,  $t_e = 8$ .

- b. use the binomial model to calculate  $p_e$ , the probability of reaching  $t_e$ ; for example if  $dr_i = 3$  and  $t_e = 5$ , Table 1 tells us that  $p_e = 10p^2q^3$ .
- c. use the model to find the probability of a false positive,  $p_{fp} = \Pr(\text{success} \mid DR(t_e) = dr_i)$ .
- d.  $p_e \times p_{fp}$  is the probability of a false positive for this value of  $dr_i$ .
- e.  $p_e \times (T_g - t_e)$  is the expected *early warning premium* for this value of  $dr_i$ .

$T_g - t_e$  is the time that remains before the deadline at the envelope boundary at  $dr_i$ ; this is why  $T_g - t_e$  is called the early warning premium. The expected early warning premium for a value of  $dr_i$  is just  $T_g - t_e$  times the probability of crossing the envelope boundary. The *mean expected early warning premium* is the mean over all values of  $dr_i$  of  $p_e(T_g - t_e)$ . We expect it to have higher values for lower slack times, because the envelope boundaries for low slack times are further from the deadline. The *mean probability of false positives* is obtained by summing  $p_e p_{fp}$  for all values of  $dr_i$  and dividing by the number of these values. We expect it to rise, also, as slack time decreases, as suggested by the contours in Figure 6.

With a table of values for the mean probability of false positives and the mean expected early warning premium, and utilities for early warning and false positives, we can make a rational decision about slack time.

## 6 Conclusion

Although we rely heavily on slack time envelopes in the Phoenix planner, we have always constructed them by heuristic criteria, and we did not know how to evaluate their performance. In this paper we showed that high performance can be achieved by hand-constructed slack time envelopes, and we presented a probabilistic model of progress, from which we derived a method for automatically constructing slack time envelopes that balance the benefits of early warnings against the costs of false positives.

Other work has been done in this area, e.g., (Miller, 1989) constructs an execution monitoring profile of acceptable ranges of sensor values for a mobile robot (this profile is also called an "envelope"). If, during plan execution, a sensor value exceeds the envelope boundaries, a reflex is triggered to adjust the robot's behavior in such a way that the sensor readings return to the acceptable range. (Sanborn and Hendler, 1988) have used monitoring and projection in a simulated robot that tries to cross a busy street. The robot has a basic street-crossing plan, but monitors oncoming traffic and predicts possible collision points which trigger reactive avoidance actions. Our contribution has been to cast the problem in

probabilistic terms and to develop a framework for evaluation. We are currently extending our work to other models of progress and different, more complex domains. A technical report covering this work in more detail is in preparation.

## Acknowledgements

This research is supported by DARPA under contract #F49620-89-C-00113, by AFOSR under the Intelligent Real-time Problem Solving Initiative, contract #AFOSR-91-0067, and by ONR under a University Research Initiative grant, ONR #N00014-86-K-0764, and by Texas Instruments Corporation. We wish to thank Eric Hansen and Cynthia Loiselle for many thoughtful comments on drafts of this paper. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

## References

- Cohen, P.R.; Greenberg, M.L.; Hart, D.M.; and Howe, A.E., 1989. Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. *AI Magazine*, 10(3):32-48.
- Coombs, C.; Dawes, R.; and Tversky, A., 1970. *Mathematical Psychology: An Elementary Introduction*. Ch. 6. The Theory of Signal Detectability. Prentice Hall.
- Hart, D.M.; Anderson, S.D.; and Cohen, P.R., 1990. Envelopes as a Vehicle for Improving the Efficiency of Plan Execution. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*. K. Sycara (Ed.) San Mateo, CA.: Morgan-Kaufmann, Inc. Pp. 71 - 76.
- McNicol, D., 1972. *A Primer of Signal Detection Theory*. George Allen and Unwin, Ltd.
- Miller, D.P., 1989. Execution Monitoring for a Mobile Robot System. In *SPIE Vol. 1196 Intelligent Control and Adaptive Systems*. Pp. 36-43.
- Sanborn, J.C.; and Hendler, J.A., 1988. Dynamic Reaction: Controlling Behavior in Dynamic Domains. *International Journal of Artificial Intelligence in Engineering*, 3(2).

## **5 Constructing an Envelope without a Model**

An abridged version of this paper will appear in the *Proceedings of the AAAI-93 Workshop on Learning Action Models*.

# **Learning a Decision Rule for Monitoring Tasks with Deadlines**

*Eric A. Hansen and Paul R. Cohen*

### *Abstract*

A real-time scheduler or planner is responsible for managing tasks with deadlines. When the time required to execute a task is uncertain, it may be useful to monitor the task to predict whether it will meet its deadline; this provides an opportunity to make adjustments or else to abandon a task that can't succeed. This paper treats monitoring a task with a deadline as a sequential decision problem. Given an explicit model of task execution time, execution cost, and payoff for meeting a deadline, an optimal decision rule for monitoring the task can be constructed using stochastic dynamic programming. If a model is not available, the same rule can be learned using temporal difference methods. These results are significant because of the importance of this decision rule in real-time computing.

## 1. Introduction

In hard real-time systems, tasks must meet deadlines and there is little or no value in completing a task after its deadline. When the time required to execute a task is uncertain, it may be useful to monitor the task as it executes so that failure to meet its deadline can be predicted as soon as possible. Anticipating failure to meet a deadline provides an opportunity to adjust: either to initiate a recovery action, or simply to abandon a failing task to conserve limited resources.

A system that monitors ongoing tasks needs a decision rule to predict whether a task will meet its deadline. The Phoenix planner uses a decision rule for this called an "envelope" (Hart, Anderson, & Cohen, 1990) which defines a range of expected performance over time. Envelopes are represented by a two-dimensional graph that plots progress toward completion of a task as a function of time; for example, in Figure 1 the shaded region represents an envelope. When the actual progress of a monitored task falls outside the envelope boundary, failure to meet the task's deadline is predicted and an appropriate action is taken.

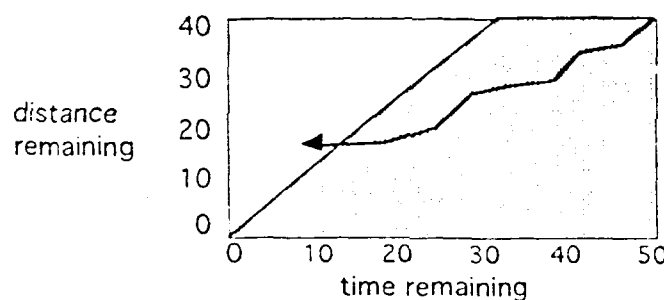


Figure 1. A Phoenix envelope. The arrow represents the trajectory of a task. When it falls outside the shaded region, a recovery action is initiated.

Monitoring progress to anticipate failure to meet a deadline is useful in other contexts besides monitoring plans. For example, real-time problem-solvers monitor their progress to determine whether to adjust their strategies to make sure a solution is ready by a deadline (Dorf & Lesser, 1988; Lesser, Pavlin, & Dorf, 1988). Dynamic schedulers for real-time operating systems monitor task execution so they can anticipate failure to meet a deadline in time to abort a failing task and conserve resources (Haben & Shin, 1990).

Despite the many contexts in which it is useful to monitor tasks with deadlines, no principled method has yet been developed for constructing optimal decision rules, that is, rules to predict whether a task will meet a deadline. The envelopes used by the Phoenix planner were handcrafted and adjusted by trial and error to get them to work.

well. Similar decision rules used in other systems have also been heuristic or constructed in an ad hoc way.

This paper describes how an optimal decision rule can be constructed. As an example, it works through the construction of a rule for the simple case in which the only option for a failing task is to abandon it. When recovery options are available a more complex decision rule is called for, but constructing it is a straightforward extension of the methods described here for constructing the rule in its simplest form.

Monitoring a task over the course of its execution requires a sequence of related decisions; each of these decisions is not whether to continue the task to the end, but whether to continue it a little longer before monitoring and reconsidering whether to continue it further. A task should be continued as long as the expected value of continuing it until the next monitoring action is greater than the expected value of abandoning it. Knowing the expected value of continuing a task until it is monitored again requires knowing the expected value of continuing it after that, and so on recursively until the deadline. In this sense, constructing an optimal decision rule is a sequential decision problem.

Section 2 describes how to use dynamic programming to unravel this recursive relationship backwards from the deadline. Using dynamic programming to construct the decision rule requires an explicit model of probable task execution time, execution cost, and the payoff for meeting the deadline. Section 3 describes machine learning techniques that can learn the decision rule if a model is not available. Section 4 discusses the generality of these results and possible extensions.

## 2. Constructing the Decision Rule Using Dynamic Programming

Dynamic programming is an optimization technique for solving problems that require making a sequence of decisions. A decision rule for monitoring task execution can be constructed using dynamic programming by assuming a task is monitored at discrete time steps; any interval of time can correspond to a time step.

The decision problem is formalized as follows. A state is represented by two variables,  $(t, d)$ , where  $t$  is a non-negative integer that represents the number of time steps remaining before the deadline and  $d$  is a non-negative integer that represents the part of the task (the "distance") that remains to be completed. (This presupposes some unit of progress in terms of which completion of a task can be measured.) The decision to continue a task or abandon it is represented by a binary decision variable,  $a \in \{\text{continue}, \text{stop}\}$ .

The amount of a task likely to be executed in one time step is represented by a set of state transition probabilities,  $P_{(t,d),(t-1,d-k)}(\text{continue})$ . In this notation, the first subscript,  $(t, d)$ , is the state at the beginning of the time step and the second subscript,  $(t-1, d-k)$ , is the state at the end of the time step, where  $k$  is the number of units of

the task completed during the time step. The argument is the action taken at the beginning of the time step.

In addition to this stochastic state-transition model, a function,  $R(t, d, a)$ , specifies the single-step payoff for action,  $a$ , taken in state,  $(t, d)$ . The execution cost per time step is  $R(t, d, \text{continue})$ ; while  $R(t, d, \text{stop})$  is the value for finishing a task by its deadline (when  $t \geq 0, d = 0$ ) or zero if the deadline is not met ( $t = 0, d \geq 1$ ).

The objective is to maximize the expected value of the sum of the single-step payoffs for the course of a task. The difficulty is that the payoff for finishing a task by its deadline is not received until the end of the task, although it must be considered in making earlier decisions about whether to continue the task. Dynamic programming solves sequential decision problems with "delayed payoff" by constructing an evaluation function that provides *secondary reinforcement*; the key idea is that expected cumulative value over the long term is maximized by choosing, at each step, the action that maximizes the evaluation function in the short term.

The evaluation function for this decision problem is expressed by the recurrence relation:

$$V(t, d) = \max \{ R(t, d, \text{stop}), E[V(t-1, d-k)] + R(t, d, \text{continue}) \}$$

Expanding the term for the expected value of the next state, this becomes:

$$V(t, d) = \max \left\{ R(t, d, \text{stop}), \sum_k [P_{(t,d),(t-1,d-k)}(\text{continue}) \cdot V(t-1, d-k)] + R(t, d, \text{continue}) \right\}$$

Dynamic programming systematically evaluates this recurrence relation to fill in a table of values,  $V(t, d)$ , that represents the evaluation function.

In the course of evaluating this recurrence relation, the optimal action in each state (whether to stop or continue) is determined. The decision rule is defined implicitly by the evaluation function, as follows.

$$\pi(t, d) = \begin{cases} \text{continue} & \text{if } V(t, d) > 0 \\ \text{stop} & \text{otherwise} \end{cases}$$

That is, continue if the expected value of continuing exceeds zero, the value for stopping. (In the terminology of dynamic programming, a decision rule is also referred to as a policy function.)

This formal notation describes the decision rule in its most general form. However it leaves the state transition probabilities and costs used to compute the rule unspecified; they are fit to the decision problem being modeled. There are two ways to obtain these probabilities and costs. One is to specify them given some exogenous knowledge about the task; the other is to learn them automatically. We will consider these in turn.

For the sake of having an example to work through in this paper, we make the following arbitrary assumptions. Each time step is regarded as a single Bernoulli trial. This makes the state transition probabilities:

$$\begin{aligned} P_{(t,d),(t-1,d-1)}(\text{continue}) &= p \\ P_{(t,d),(t-1,d)}(\text{continue}) &= 1-p \end{aligned}$$

where  $p$  is the probability of "success" in a Bernoulli trial. Completing a task of size  $d$  is equivalent to succeeding in  $d$  Bernoulli trials, and completing the task of size  $d$  in time  $t$  is equivalent to succeeding in  $d$  out of  $t$  trials. So task execution time is binomially distributed with a mean and variance proportional to time.

Because a binomial distribution is approximately normal when  $tp(1-p) \geq 10$ , and because it seems likely that in many cases task execution time is approximately normally distributed, this is a plausible model. The mean and variance of the binomial probability function can be fit to the actual mean and variance of task execution time by choosing the value of  $p$  so that the equation,

$$p = 1 - \text{Variance}(\text{execution time}) / \text{Mean}(\text{execution time})$$

is satisfied, and by choosing the scale of the distance step so that the mean proportion of a task executed in one time step is  $p \cdot (\text{distance step})$ .

As a payoff function, we assume:

$$\begin{aligned} R(0,d,\text{stop}) &= 0 \quad \text{for } d \geq 1 \\ R(t,0,\text{stop}) &= R \quad \text{for } t \geq 0 \\ R(t,d,\text{continue}) &= E \quad \text{for } t,d \geq 1 \end{aligned}$$

where  $R$  is the reward for finishing a task by its deadline and  $E$  is the execution cost per time step. Like the binomial state-transition model, this parameterized payoff function is very general and likely to fit many situations. However it too is only an example; any payoff function could be used.

Given these state-transition probabilities and payoffs, the evaluation function for this decision problem has the following simple recursive definition:

$$\begin{aligned} V(0,d) &= 0 \quad \text{for } d \geq 1 \\ V(t,0) &= R \quad \text{for } t \geq 0 \\ V(t,d) &= \max\{0, p \cdot V(t-1,d-1) + (1-p) \cdot V(t-1,d) + E\} \quad \text{for } t,d \geq 1 \end{aligned}$$

The evaluation function for parameter values,  $p = 0.5$ ,  $F = 100$  and  $E = -1$ , is shown in Figure 2.

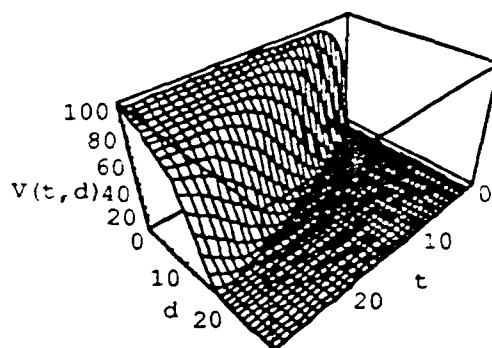


Figure 2. Evaluation function computed by dynamic programming.

The decision rule is represented implicitly by the evaluation function, since a task is continued as long as the expected value of the current state is greater than zero.

In the following graph the decision rule is projected onto two dimensions and extended out to a starting time of 200 before the deadline; this shows that for  $d \geq 50$  it is not worth even beginning the task because the expected cost of completing the task is greater than the potential reward for finishing it.

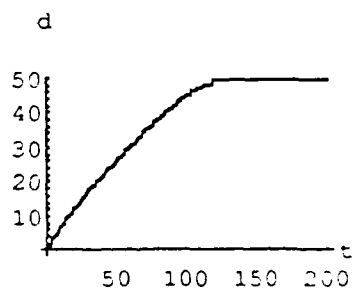


Figure 3. Representation of decision rule computed by dynamic programming.

If the trajectory of a task goes above the line, the task is abandoned.

The shape of this decision rule is strikingly similar to the shape of the envelopes constructed by trial and error and used by the Phoenix planner.

### 3. Learning the Decision Rule Using Temporal Difference Methods

When an accurate model of the state transition probabilities and costs is available, an optimal decision rule can be computed off-line using dynamic programming. When an accurate model is not available, however, there are still machine learning methods that can gradually adapt a decision rule until it converges to one that is optimal. Called temporal difference (TD) methods (Barto, Sutton, & Watkins, 1990), they solve the so-called "temporal credit-assignment problem" inherent in sequential decision problems with delayed payoff by, in effect, approximating dynamic programming.

It can be shown that the recurrence relation for an evaluation function constructed by dynamic programming only holds true for a decision rule (i.e., policy) that is optimal. This provides the basis for TD methods. TD methods learn an evaluation function in addition to a decision rule. In our monitoring example, the recurrence relation that defines the evaluation function is

$$V(t, d) = E[V(t-1, d-k)] + R(t, d, \text{continue})$$

Any measured difference during training between the values of the two sides of the equation is treated as an "error" that is used to adjust the decision rule. The TD error measured after each time step is:

$$V(t, d) - V(t-1, d-k) - R(t, d, \text{continue})$$

The difference in this definition of TD error and the recurrence relation for the evaluation function is that the expected value,  $E[V(t-1, d-k)]$ , is replaced by  $V(t-1, d-k)$ . This is necessary because the expected value cannot be computed without a model of the state-transition probabilities. However TD training works because the learned value of each state regresses towards the weighted average of the values of its successor states, where the weightings reflect the conditional probabilities of the successor states. So in the limit, the value of each state converges to the expected value of its successor states plus the single-step payoff, and hence converges to the expected cumulative value.

By adapting the decision rule to minimize the TD error, an optimal decision rule is gradually learned along with an evaluation function. Adjusting the decision rule changes the evaluation function, which in turn serves as feedback to the learning algorithm for continued adjustment of the decision rule; the two are adapted simultaneously. In most cases of TD learning, the decision rule and evaluation function must be represented separately. However this example is particularly straightforward because the simple relationship between the two ---the decision rule is defined by a simple threshold on the evaluation function--- makes it possible to represent both by the same function.

A complication is that learning takes place only as long as a task is continued, and so only inside the "boundary" of the decision rule. If this boundary is inadvertently set too conservatively, it cannot be unlearned unless a task is occasionally continued from a state outside the boundary to see what happens. This is characteristic of trial-and-error learning; occasionally actions that appear suboptimal must be taken so that the relative merits of actions can be assessed. This is managed by including a stochastic element in the decision rule, for example:

$$\pi(t, d) = \begin{cases} \text{continue if } (V(t, d) + (\text{random}(2, 0) - 1)) > 0 \\ \text{stop otherwise} \end{cases}$$

The decision to use TD methods for training is independent of the decision about what function representation and learning algorithm to use. We show this by describing two

different representations for the evaluation function and two different learning algorithms.

### 3.1 Table Representation and Linear Update Rule

If we represent the evaluation function by a two-dimensional table, as we have for dynamic programming, then the values in the table can be adjusted by the following learning rule:

$$V(t,d) := V(t,d) + \alpha \cdot \text{error} \cdot V(t,d)$$

This learning rule increments or decrements the value of the current state by an amount proportional to the TD error, defined as:

$$\text{error} = R(t,d,\text{continue}) + V(t-1,d-k) - V(t,d)$$

as well as proportional to a learning rate,  $\alpha$  (in this example, set to 0.1). This linear learning rule minimizes the TD error by gradient descent.

A training regimen consists of repeatedly starting a task from a random state,  $(t,d)$ , and continuing it until it finishes or is abandoned; each task counts as a learning trial. For the purpose of generating a learning curve, performance is measured by comparing the evaluation function computed by stochastic dynamic programming to the table of values learned by TD training and measuring the mean square error. This comparison gives rise to the learning curve shown in Figure 4.

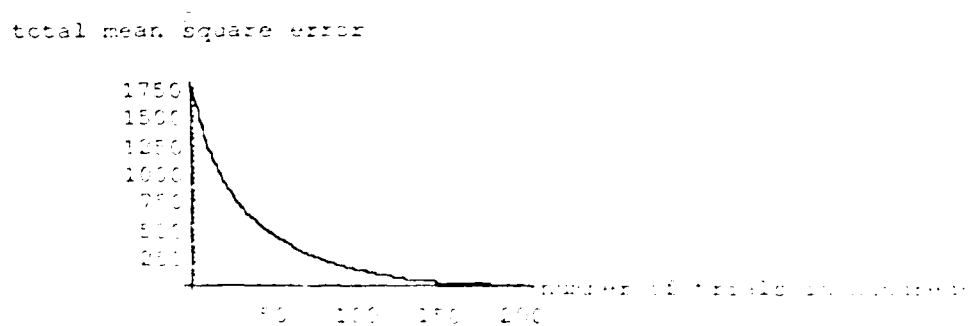


Figure 4. Learning curve for TD training, using a table representation and linear update rule.

The learned evaluation function, shown in Figure 5, is nearly identical to the optimal evaluation function computed by dynamic programming.

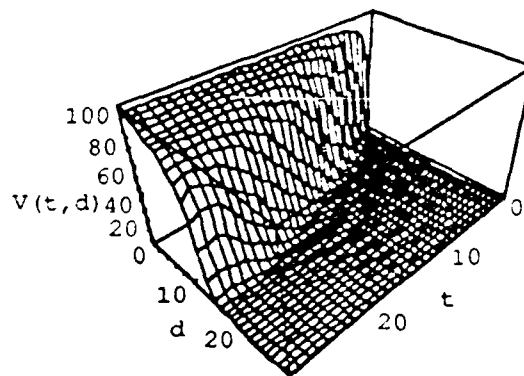


Figure 5. Evaluation function learned by TD training, using a table representation and linear update rule.

### 3.2 Connectionist Representation and Error Backpropagation Rule

The problem with representing the evaluation function by a table is that it requires  $O(n^2)$  storage, where  $n$  is the number of time steps from the start of the task to its deadline. A more compact function representation would be better, although it still must be capable of representing a nonlinear function. One possibility is a feedforward connectionist network trained by the error backpropagation rule. The simplest network possible for this problem is a single neuron with two inputs and a sigmoid activation function. It corresponds to the formula

$$V(t, d) = \frac{1}{1 + e^{-(w_1 t + w_2 d + w_3)}}$$

where  $w_1$ ,  $w_2$ , and  $w_3$  are the learned weights. This simple representation turns out to work surprisingly well. Trained by temporal differences using backpropagation, the learning curve in Figure 6 illustrates that it converges many times faster than when a table representation and linear update rule are used.

total mean square error

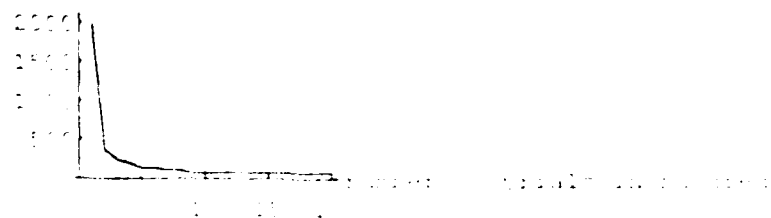
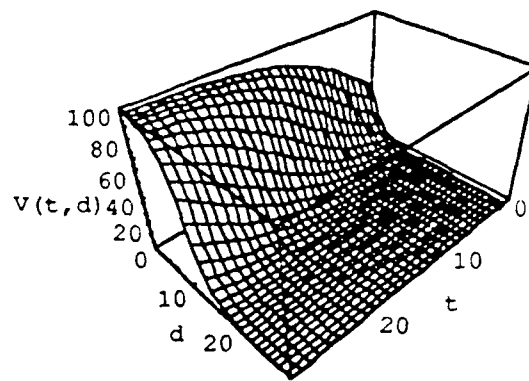


Figure 6. Learning curve of the single neuron network trained on



**Figure 7.** Evaluation function learned by TD training of a single neuron by error backpropagation.

The reason convergence is faster is that the parameterized function represented by the neuron makes generalization possible; in addition, it is able to closely approximate the optimal evaluation function. The learned evaluation function is shown in Figure 7.

Besides being more space-efficient, a connectionist network also has the advantage of being able to represent a continuous evaluation function, instead of the discrete function presupposed by a table representation. It allows for generalization as well because the possible input values are not limited by the size of the table.

#### 4. Discussion

This paper treats monitoring tasks with deadlines as a sequential decision problem, which makes available a class of methods based on dynamic programming for constructing a decision rule for monitoring. When an explicit model of the state-transition probabilities and costs is available, the rule can be constructed off-line using stochastic dynamic programming. Otherwise it can be learned on-line using TD methods that approximate dynamic programming.

It makes sense to construct a decision rule such as the one described in this paper for tasks that are repeated many times or for a class of tasks with the same behavior. This allows the rule to be learned, if TD methods are relied on; or for statistics to be gathered to characterize a probability and cost model, if dynamic programming is relied on. However if a model is known beforehand, or can be estimated, a decision rule can also be constructed for a task that executes only once.

The time complexity of the dynamic programming algorithm is  $O(n^2)$ , where  $n$  is the number of time steps from the start of the task to its deadline; however the decision rule may be compiled once and reused for subsequent tasks. The time complexity of TD learning,  $O(n)$ , is mitigated by the possibility of turning learning off and on. The space

overhead of representing an evaluation function by a table is avoidable by using a more compact function representation, such as a connectionist network.

Besides the fact that the approach described in this paper is not computationally intensive, it has other advantages. It is conceptually simple. The decision rule it constructs is optimal, or converges to the optimal in the case of TD learning. It works no matter what probability model characterizes the execution time of a task and no matter what cost model applies, and so is extremely general. Finally, it works even when no model of the state transition probabilities and costs is available, although a model can be taken advantage of.

These results can be extended in a couple obvious ways. The first is to factor in a cost for monitoring. In this paper we assume monitoring has no cost, or its cost is negligible. This allows monitoring to be nearly continuous, in effect, for a task to be monitored each time step. Others who have developed similar decision rules have also assumed the cost of monitoring is negligible. However in some cases the cost of monitoring may be significant, so in another paper we show how this cost can be factored in (forthcoming). Once again we use dynamic programming and TD methods to develop optimal monitoring strategies.

The second way in which this work can be extended is to make the decision rule more complicated. In this paper we analyzed a simple example in which the only alternative to continuing a task is to abandon it. But recovery options may be available as well. A dynamic scheduler for a real-time operating system is unlikely to have recovery options available, but an AI planner or problem-solver is almost certain to have them (Lesser, Pavlin & Durfee, 1988; Howe, 1992). The way to handle the more complicated decision problem this poses is to regard each recovery option as a separate task characterized by its own probability model and cost model; so at any point the expected value of the option can be computed. Then instead of choosing between two options, either continuing a task or abandoning it, the choice includes the recovery options as well. The rule is simply to choose the option with the highest expected value.

## Acknowledgments

This research is supported by the Defense Advanced Research Projects Agency under contract #F49620-89-C-00113; by the Air Force Office of Scientific Research under the Intelligent Real-time Problem Solving Initiative, contract #AFOSR-91-0067; and by Augmentation Award for Science and Engineering Research Training, PR No. C-2-2675. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

## References

- Barto, A.G., Sutton, R.S., & Watkins, C.J.C.H. 1990. Learning and sequential decision making. In *Learning and Computational Neuroscience: Foundations of Adaptive Networks*. M. Gabriel and J. W. Moore (Eds.), MIT Press, Cambridge, MA. Pp. 539-602.
- Durfee, E. & Lesser, V. 1988. Planning to meet deadlines in a blackboard-based problem solver. In *Hard Real-Time Systems*. J. Stankovic and K. Ramamrithan (Eds.), IEEE Computer Society Press, Los Alamitos, CA. Pp. 595-608.
- Haben, D. & Shin, K. 1990. Application of real-time monitoring to scheduling tasks with random execution times. In *IEEE Transactions on Software Engineering* 16(2): 1374-1389.
- Hart, D.M., Anderson, S.D., & Cohen, P.R. 1990. Envelopes as a vehicle for improving the efficiency of plan execution. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*. K. Sycara (Ed.), San Mateo, CA: Morgan-Kaufman, Inc. Pp. 71-76.
- Howe, A.E. 1992. Analyzing failure recovery to improve planner design. In *Proceedings AAAI-92*. Pp. 387-392.
- Lesser, V., Pavlin, J. & Durfee, E. 1988. Approximate processing in real-time problem-solving. In *AI Magazine* 9(1): 49-61.

## 6. Building Causal Models of Planner Behavior using *Path Analysis*

This paper appeared in the *Proceedings of the First International Conference on AI Planning Systems*.

# Predicting and Explaining Success and Task Duration in the Phoenix Planner

David M. Hart and Paul R. Cohen

## Abstract

Phoenix is a multi-agent planning system that fights simulated forest-fires. In this paper we describe an experiment with Phoenix in which we uncover factors that affect the planner's behavior and test predictions about the planner's robustness against variations in some of these factors. We also introduce a technique—path analysis—for constructing and testing causal explanations of the planner's behavior.

## 1 INTRODUCTION

It is difficult to predict or even explain the behavior of any but the simplest AI programs. A program will solve one problem readily, but make a complete hash of an apparently similar problem. For example, our Phoenix planner, which fights simulated forest fires, will contain one fire in a matter of hours but fail to contain another under very similar conditions. We therefore hesitate to claim that the Phoenix planner "works." The claim would not be very informative, anyway: we would much rather be able to predict and explain Phoenix's behavior in a wide range of conditions (Cohen 1991). In this paper we describe an experiment with Phoenix in which we uncover factors that affect the planner's behavior and test predictions about the planner's robustness against variations in some factors. We also introduce a technique—path analysis—for constructing and testing causal explanations of the planner's behavior. Our results are specific to the Phoenix planner and will not necessarily generalize to other planners or environments, but our techniques are general and should enable others to derive comparable results for themselves.

In overview, Section 2 introduces the Phoenix planner; Section 3 describes an experiment in which we identify factors that probably influence the planner's behavior; and Section 4 discusses results and one sense in which the

planner works "as designed." But these results leave much unexplained: although Section 4 identifies some factors that affect the success and the duration of fire-fighting episodes, it does not explain how these factors interact. Section 5 shows how correlations among the factors that affect behavior can be decomposed to test causal models that include these factors.

## 2 PHOENIX OVERVIEW

Phoenix is a multi-agent planning system that fights simulated forest-fires. The simulation uses terrain, elevation, and feature data from Yellowstone National Park and a model of fire spread from the National Wildlife Coordinating Group Fireline Handbook (National Wildlife Coordinating Group 1985). The spread of fires is influenced by wind and moisture conditions, changes in elevation and ground cover, and is impeded by natural and man-made boundaries such as rivers, roads, and fireline. The Fireline Handbook also prescribes many of the characteristics of our firefighting agents, such as rates of movement and effectiveness of various firefighting techniques. For example, the rate at which bulldozers dig fireline varies with the terrain. Phoenix is a real-time simulation environment—Phoenix agents must think and act as the fire spreads. Thus, if it takes too long to decide on a course of action, or if the environment changes while a decision is being made, a plan is likely to fail.

One Phoenix agent, the Fireboss, coordinates the firefighting activities of all field agents, such as bulldozers and watchtowers. The Fireboss is essentially a thinking agent,<sup>1</sup> using reports from field agents to form and maintain a global assessment of the world. Based on these reports (e.g., fire sightings, position updates, line

<sup>1</sup> Though it has the same architecture as other agents, it has few sensors or effectors and is immobile. For a detailed description of the Phoenix agent architecture and planning mechanisms see Cohen et al. 1989.

progress), it selects and instantiates fire-fighting plans and directs field agents in the execution of plan subtasks.

A new fire is typically spotted by a watchtower, which reports observed fire size and location to the Fireboss. With this information, the Fireboss selects an appropriate fire-fighting plan from its plan library. Typically these plans dispatch bulldozer agents to the fire to dig fireline. An important first step in each of the three plans in the experiment described below is to decide where fireline should be dug. The Fireboss projects the spread of the fire based on prevailing weather conditions, then considers the number of available bulldozers and the proximity of natural boundaries. It projects a bounding polygon of fireline to be dug and assigns segments to bulldozers based on a periodically updated assessment of which segments will be reached by the spreading fire soonest. Because there are usually many more segments than bulldozers, each bulldozer digs multiple segments. The Fireboss assigns segments to bulldozers one at a time, then waits for each bulldozer to report that it has completed its segment before assigning another. This ensures that segment assignment incorporates the most up-to-date information about overall progress and changes in the prevailing conditions.

Once a plan is set into motion, any number of problems might arise that require the Fireboss's intervention. The types of problems and mechanisms for handling them are described in Howe & Cohen 1990, but one is of particular interest here: As bulldozers build fireline, the Fireboss compares their progress to expected progress.<sup>2</sup> If their actual progress falls too far below expectations, a plan failure occurs, and (under the experiment scenario described here) a new plan is generated. The new plan uses the same bulldozers to fight the fire and exploits any fireline that has already been dug. We call this error recovery method *replanning*. Phoenix is built to be an adaptable planning system that can recover from plan failures (Howe & Cohen 1990). Although it has many failure-recovery methods, replanning is the focus of the experiment described in the next section.

### 3 IDENTIFYING THE FACTORS THAT AFFECT PERFORMANCE

<sup>2</sup> Expectations about progress are stored in *envelopes*. Envelopes represent the range of acceptable progress, given the knowledge used to construct the plan. If actual progress falls outside this range, and envelope violation occurs, invoking error recovery mechanisms (Cohen, Hart & St. Amant 1992, Hart, Anderson & Cohen 1990).

We designed an experiment with two purposes. A *confirmatory* purpose was to test predictions that the planner's performance is sensitive to some environmental conditions but not others.<sup>3</sup> In particular, we expected performance to degrade when we change a fundamental relationship between the planner and its environment—the amount of time the planner is allowed to think relative to the rate at which the environment changes—and not be sensitive to common dynamics in the environment such as weather, and particularly, wind speed. We tested two specific predictions: 1) that performance would not degrade or would degrade gracefully as wind speed increased; and 2) that the planner would not be robust to changes in the Fireboss's thinking speed due to a bottleneck problem described below. An *exploratory* purpose of the experiment was to identify the factors in the Fireboss architecture and Phoenix environment that most affected the planner's behavior, leading to the causal model developed in Section 5.

The Fireboss must select plans, instantiate them, dispatch agents and monitor their progress, and respond to plan failures as the fire burns. The rate at which the Fireboss thinks is determined by a parameter called the *Real Time Knob*. By adjusting the Real Time Knob we allow more or less simulation time to elapse per unit CPU time, effectively adjusting the speed at which the Fireboss thinks relative to the rate at which the environment changes.

The Fireboss services bulldozer requests for assignments, providing each bulldozer with a task directive for each new fireline segment it builds. The Fireboss can become a bottleneck when the arrival rate of bulldozer task requests is high or when its thinking speed is slowed by adjusting the Real Time Knob. This bottleneck sometimes causes the overall digging rate to fall below that required to complete the fireline polygon before the fire reaches it, which causes replanning (see Section 2). In the worst case, a Fireboss bottleneck can cause a thrashing effect in which plan failures occur repeatedly because the Fireboss can't assign bulldozers during replanning fast enough to keep the overall digging rate at effective levels. We designed our experiment to explore the effects of this bottleneck on system performance and to confirm our prediction that performance would vary in proportion to the manipulation of thinking speed. Because the current design of the Fireboss is not sensitive to changes in thinking speed, we expect it

<sup>3</sup> The term "planner" here refers collectively to all Phoenix agents, as distinct from the Fireboss agent.

to take longer to fight fires and to fail more often to contain them as thinking speed slows.

In contrast, we expect Phoenix to be able to fight fires at different wind speeds. It might take longer and sacrifice more area burned at high wind speeds, but we expect this effect to be proportional as wind speed increases and we expect Phoenix to succeed equally often at a range of wind speeds, since it was designed to do so.

### 3.1 EXPERIMENT DESIGN

We created a straightforward fire fighting scenario that controlled for many of the variables known to affect the planner's performance. In each trial, one fire of a known initial size was set at the same location (an area with no natural boundaries) at the same time (relative to the start of the simulation). Four bulldozers were used to fight it. The wind's speed and direction were set initially and not varied during the trial. Thus, in each trial, the Fireboss receives the same fire report, chooses a fire-fighting plan, and dispatches the bulldozers to implement it. A trial ends when the bulldozers have successfully surrounded the fire or after 120 hours without success.

The experiment's first dependent variable then is Success, which is true if the fire is contained, and false otherwise. A second dependent variable is shutdown time (SD), the time at which the trial was stopped. For successful trials, shutdown time tells us how long it took to contain the fire.<sup>4</sup>

Two independent variables were wind speed (WS) and the setting of the Fireboss's Real Time Knob (RTK). A third variable, the first plan chosen by the Fireboss in a trial (FPLAN), varied randomly between trials. It was not expected to influence performance, but because it did, we treat it here as an independent variable.

**WS:** The settings of WS in the experiment were 3, 6, and 9 kilometers per hour. As wind speed increases, fire spreads more quickly in all directions, and most quickly downwind. The Fireboss compensates for higher values of wind speed by directing bulldozers to build fireline further from the fire.

**RTK:** The default setting of RTK for Phoenix agents allows them to execute 1 CPU second of Lisp code for every 5 minutes that elapses in the simulation. We varied the Fireboss's RTK setting in different trials (leaving the set-

tings for all other agents at the default). We started at a ratio of 1 simulation-minute/cpu-second, a thinking speed 5 times as fast as the default, and varied the setting over values of 1, 3, 5, 7, 9, 11, and 15 simulation-minutes/cpu-second. These values range from 5 times the normal speed at a setting of 1 down to one-third the normal speed at 15. The values of RTK reported here are rescaled. The normal thinking speed (5) has been set to RTK=1, and the other settings are relative to normal. The scaled values (in order of increasing thinking speed) are .33, .45, .56, .71, 1, 1.67, and 5. RTK was set at the start of each trial and held constant throughout.

**FPLAN:** The Fireboss randomly selects one of three plans as its first plan in each trial. The plans differ mainly in the way they project fire spread and decide where to dig fireline. SHELL is aggressive, assuming an optimistic combination of low fire spread and fast progress on the part of bulldozers. MODEL is conservative in its expectations, assuming a high rate of spread and a lower rate of progress. The third, MBIA, generally makes an assessment intermediate with respect to the others.<sup>5</sup> When replanning is necessary, the Fireboss again chooses randomly from among the same three plans.<sup>6</sup>

We adopted a basic factorial design, systematically varying the values of WS and RTK. Because we had not anticipated a significant effect of FPLAN, we allowed it to vary randomly.

## 4 RESULTS FOR SUCCESS RATE AND SHUTDOWN TIME

We collected data for 343 trials, of which 215 succeeded and 128 failed, for an overall success rate of 63%. Tables 1a-c break down successes and failures for each setting of the independent variables RTK, WS, and FPLAN. Column S in these tables is the number of Successes, F is the number of Failures, and Tot is the total number of trials. Certain trends emerge in these data that confirm our earlier predictions. For example, in Table 1a, the success rate improves steadily as the thinking speed of the Fireboss

<sup>4</sup> Several other dependent variables were measured, most notably Area Burned. However, using Area Burned to assess performance requires stricter experimental controls over such factors as choice of fire-fighting plan than were used here.

<sup>5</sup> The first plan of this variety developed in Phoenix was called Multiple-Bulldozer-Indirect-Attack, or MBIA, which signified a coordination of bulldozers working at some distance from the fire on fireline segments determined by the Fireboss's projections. SHELL is a variant of MBIA that builds a tighter shell of fireline, thus reducing the cost of forest burned. MODEL is another variant of MBIA that applies an analytical model of fire projection (Cohen 1988). It makes conservative projections at the default parameters used in this experiment.

<sup>6</sup> The same high level plans can be used in the initial attack on a fire and on subsequent fires. When used in replanning, a plan is adapted to take advantage of any fireline that has already been dug near the fire. It is also based on updated conditions such as the current size and shape of the fire.

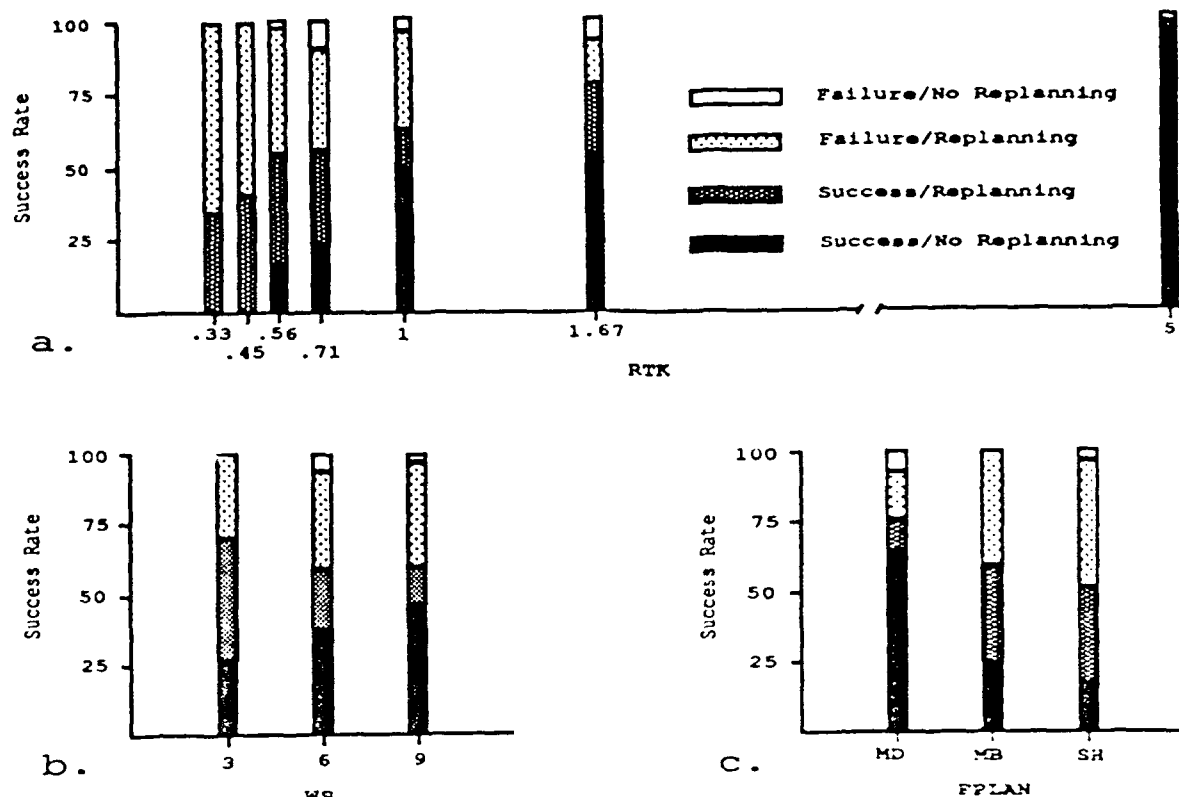


Figure 1: Successes by a) Real Time Knob, b) Wind Speed, and c) First Plan Tried

increases. However, other patterns are less clear, such as the differences for each setting of WS in Table 1b. How do we know if these values are significantly different? For a categorical dependent variable such as Success (which has only two possible values), a chi-square test ( $\chi^2$ ) will determine whether the observed pattern is statistically significant.

Figures 1a-c show the success rates for each setting of each independent variable. The table categories Success and Failure are broken down further into those trials which did not replan and those that did.

#### 4.1 EFFECT OF INDEPENDENT VARIABLES ON SUCCESS

Table 1a shows successes by the independent variable RTK. A chi-square test on the Success-Failure  $\times$  RTK contingency table in Table 1a is highly significant ( $\chi^2(6) = 49.081$ ,  $p < 0.001$ ), indicating that RTK strongly influences the relative frequency of successes and failures. At the fastest thinking speed for the Fireboss, RTK=5, the success rate is 98%, but at the slowest rate, RTK=.33, the success rate is only 33%. Figure 1a shows graphically

that as RTK goes down (i.e., thinking speed decreases) the success rate declines. At RTK=1, the default setting, 63% of the trials were successful. Note how rapidly the success of the initial plan decreases—for RTK  $\leq .45$ , no trial succeeds without replanning. However, the overall success rate declines more slowly as replanning is used to recover from the bottleneck effect described in Section 3. If we compare the rate of success without replanning to that with replanning in Figure 1a, we see that replanning buffers the Phoenix planner, allowing it to absorb the effect of changes in Fireboss RTK without failing. This effect is statistically highly significant.

Table 1a: Trials Partitioned by Real Time Knob.

RTK	S	F	Tot
.33	10	20	30
.45	14	19	33
.56	22	18	40
.71	14	12	26
1	27	36	63
1.67	34	10	44
5	47	1	48

Table 1b shows successes by wind speed. The small differences in success are marginal ( $\chi^2(2) = 5.354$ ,  $p < 0.069$ ), as we predicted in Section 3. Figure 1b shows a curious trend—as WS increases, the success rate for the first plan goes up, while the success rate in trials involving replanning diminishes. The increase in success rate for the first plan occurs because as WS increases, Phoenix overestimates the growth of the fire and plans a more conservative containing fireline.

Table 1b: Trials Partitioned by Wind Speed.

WS	S	F	Tot
3	85	35	120
6	67	50	117
9	63	43	106

Table 1c shows successes by first plan tried. Differences in success are highly significant ( $\chi^2(2) = 16.183$ ,  $p < 0.001$ ), which we had not expected when designing the experiment. As shown in Figure 1c, SHELL has a very low success rate without replanning, reflecting its aggressive character, while the conservative MODEL has an initial success rate of 65%. MBIA's initial success rate is slightly better than SHELL's (though the difference is not statistically significant).

Table 1c: Trials Partitioned by First Plan Tried.

FPLAN	S	F	Tot
shell	69	62	131
mbia	48	35	83
model	98	31	129

#### 4.2 EFFECT OF RTK ON SHUTDOWN TIME

Figure 2 shows the effect of RTK on the dependent variable Shutdown time (SD). The interesting aspect of this behavior is the transition at  $RTK=1$ . SD increases gradually between  $RTK=5$  and 1, and the 95% confidence intervals around the mean values overlap. Below 1, however, the slope changes markedly and the confidence intervals are almost disjoint from those for values above 1. This shift in slope and value range for SD suggests a threshold effect in Phoenix as the Fireboss's thinking speed is reduced below the normal setting of RTK. The cost of resources in Phoenix is proportional to the time spent fighting fires, so a threshold effect such as this represents a significant discontinuity in the cost function for resources used. For this reason we pursued the cause(s) of this discontinuity by modeling the effects of the indepen-

dent variables on several key endogenous variables,<sup>7</sup> and through them on SD, with the intent of building a causal model of the influences on SD.

#### 5 INFLUENCE OF ENDOGENOUS VARIABLES ON SHUTDOWN TIME

We measured about 40 endogenous variables in the experiment described above, but three are of particular interest in this analysis: the amount of fireline built by the bulldozers (FB), the number of fire-fighting plans tried by the Fireboss for a given trial (#PLANS), and the overall utilization of the Fireboss's thinking resources (OVUT).

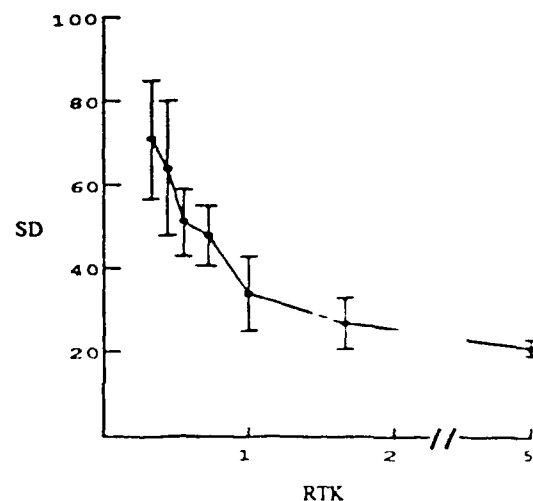


Figure 2: Mean Shutdown Time (in Hours) by Real Time Knob. Error Bars Show 95% Confidence Intervals.

**FB:** The value of this variable is the amount of fireline actually built at the end of the trial. FB sets a lower limit on SD, because bulldozers have a maximum rate at which they can dig. Thus, when the Fireboss is thinking at the fastest speed and servicing bulldozers with little wait time, SD will be primarily determined by how much fireline must be built.

**#PLANS:** When a trial ran to completion without replanning, #PLANS was set to 1. Each time the Fireboss replanned, #PLANS was incremented. #PLANS is an important indicator of the level of difficulty the planner has fighting a particular fire. It also directly affects FB. As described in Section 2, replanning involves projecting a

<sup>7</sup> A variable is called endogenous if it is influenced by independent variables and influences, perhaps indirectly through other endogenous variables, dependent variables.

new polygon for the bulldozers to dig. Typically the new polygon is larger than the previous one, because the fire has now spread to a point where the old one is too close to the fire. Thus, the amount of fireline to be dug tends to increase with the number of replanning episodes.

**OVUT:** This variable, overall utilization, is the ratio of the time the Fireboss spends thinking to the total duration of a trial. Thinking activities include monitoring the environment and agents' activities, deciding where fireline should be dug, and coordinating agents' tasks (Cohen et al. 1989). The Fireboss is sometimes idle, having done everything on its agenda, and so it waits until a message arrives from a field agent or enough time passes that another action becomes eligible. We expected to see OVUT increase as RTK decreases; that is, as the Fireboss's thinking speed slows down, it requires a greater and greater proportion of the time available to do the cognitive work required by the scenario. Replanning only adds to the Fireboss's cognitive workload.

### 5.1 REGRESSION ANALYSIS

Having identified these variables, we set about quantifying their effects using multiple regression.<sup>8</sup> We regressed SD on WS, RTK, FPLAN, OVUT, #PLANS and FB. These factors accounted for 76% of the variance in SD. Standardized beta coefficients are often cited as measures of the relative influence of factors; in Table 2a they tell us that FB has the largest influence on SD (beta = .759), with RTK and OVUT following close behind. But if the beta's represent the strength of influence, they are surprising. OVUT has a negative influence on SD, which is counterintuitive and appears to contradict the positive correlation (.42) between them in Table 2b. WS and #PLANS have virtually no influence on SD, even though #PLANS is strongly correlated with SD (.718). And although WS is essentially uncorrelated with SD (-.053), it is correlated with FB (.363), which in turn is strongly correlated with SD (.755). Finally, WS and RTK are correlated in Table 2b (.282), which seems impossible given that they were varied systematically. In short, the regression analysis and the correlation matrix contain counterintuitive entries. We will see this is because regression is based on an implicit model, one that almost certainly does not correspond to the structure of Phoenix.

<sup>8</sup> Multiple regression builds a linear model of the effects of any number of *X* variables on a continuous variable *Y*, which in this case is SD. It fits a hyperplane to the data in an *n*-dimensional space using the least-squares method, where *n* = the number of *X* variables + 1.

Table 2a: Regression For Y: SD on X's: WS, RTK, FPLAN, OVUT, #PLANS, FB

	B	Beta	t statistic of B
WS	-2.564	-0.261	-5.334 p < .001
RTK	-8.057	-0.580	-6.503 p < .001
FPLAN	.968	.035	.827 p < .283
OVUT	-.347	-.438	-4.879 p < .001
#PLANS	3.411	.115	1.742 p < .088
FB	.002	.759	11.641 p < .001

Table 2b: Correlation Coefficients

	WS	RTK	FPLAN	OVUT	#PLANS	FB
WS	1.000					
RTK	.282	1.000				
FPLAN	.117	.151	1.000			
OVUT	-.257	-.913	-.016	1.000		
#PLANS	-.183	-.409	-.432	.379	1.000	
FB	.363	-.249	-.088	.288	.658	1.000
SD	-.053	-.484	-.193	.420	.718	.755

### 5.2 PATH ANALYSIS

A technique called *path analysis* (Asher 1983, Li 1975) lets us view correlation coefficients of the variables in Table 2b as sums of hypothesized influences among factors. Consider the surprising result that wind speed (WS) is essentially uncorrelated with shut-down time (SD). We expected WS to have two possible effects on SD:

Effect 1. If WS increases then the fire burns faster, and this means more fireline must be built (i.e., FB increases), which will take longer. Therefore increasing WS should increase SD.

Effect 2. For high wind speeds, if a fire isn't contained relatively quickly, then it might not be contained at all. For example, if a fire has been burning for 60 hours or more, and WS = 3, then the probability of the fire being eventually contained is .375. But if WS = 6, the probability of eventually containing an old fire is only .2, and if WS = 9, the probability drops to .13. We measured SD for successful trials only, because, by definition, an unsuccessful trial is one that exceeds a specified SD without containing the fires. But successful containment of old fires is relatively unlikely at higher wind speeds, so as WS increases, we see fewer older fires contained, thus fewer high values of SD. This leads us to expect a negative correlation between WS and SD. Note that this correlation represents an effect of missing data, not a true negative causal relationship between WS and SD.

One of the measures produced by multiple regression is  $R^2$ , which is the percentage of variance accounted for by the linear model.

Path analysis enables us to test a model in which the correlation  $r_{WS SD}$  is composed of Effect 1 and Effect 2, which cancel each other out. Consider, for example, the path diagram in Figure 3. It shows WS positively influencing the amount of fireline that gets built (FB), and FB positively influencing SD (we will shortly describe how the numbers are derived). This path,  $WS \rightarrow FB \rightarrow SD$ , corresponds to Effect 1, above, and is called an *indirect* effect of WS on SD, mediated by FB. At the same time, WS *directly* and negatively influences SD on the path  $WS \rightarrow SD$ , corresponding to Effect 2. Figure 3 shows the strength of  $WS \rightarrow SD$  is  $-.377$ . The rules of path analysis dictate that the strength of  $WS \rightarrow FB \rightarrow SD$  is the product of the strengths of the constituent links,  $WS \rightarrow FB$  and  $FB \rightarrow SD$ , that is,  $(.363)(.892) = .328$ . The estimate of the correlation between WS and SD,  $\hat{r}_{WS SD}$ , is obtained by summing the direct and indirect effects, that is,  $.328 - .377 = -.053$ . This is the sum of all legal ways for WS to influence SD given the structure in Figure 3. For the model in Figure 3,  $\hat{r}_{WS SD} = r_{WS SD}$ , but this doesn't happen in general.

Thus we decompose the correlation  $r_{WS SD}$  into two additive effects: WS increases FB as expected and decreases SD (spuriously, as noted above) as expected, and these effects cancel.

Path analysis involves three steps:

- 1) Propose a *path model* (such as the one in Figure 3). The model represents causal influences with directed arrows (e.g.,  $FB \rightarrow SD$ ) and correlations with undirected links (see Figure 4a).
- 2) Derive *path coefficients* (such as  $-.377$ ,  $.363$  and  $.892$ ). The magnitude of a path coefficient is interpreted as a measure of causal influence.
- 3) Estimate the strength of the relationship between two factors (such as WS and SD) by multiplying path coefficients along paths between the factors and summing the products over all legal paths between the factors.

Step 3 is entirely algorithmic given some simple rules (described below) that define legal paths. Step 2 involves some judgment because some models allow multiple ways to derive one or more path coefficients. A model is a concise statement of hypothesized causal influences among factors, and the space of models grows combinatorially with the number of factors, so step 1, proposing a model,

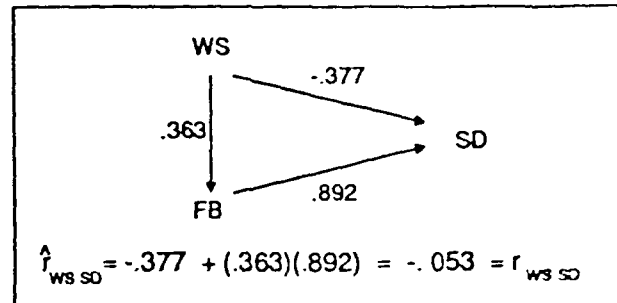


Figure 3: A Simple Path Diagram Showing Three Variables and Their Influences.

is apt to benefit from knowledge about the system we are modeling.<sup>9</sup>

All three steps will be clearer if we briefly describe the relationship between multiple linear regression and path analysis. They are basically the same thing: both derive path coefficients for a model. The difference is simply that one particular model is implicit in multiple regression. Consider an elaboration of Figure 3, in which we add the RTK as an additional causal influence on SD. Figure 4a shows the implicit model fit by multiple regression, and Figure 4b shows a model that we think is a better representation of what is actually going on in Phoenix.

The regression model assumes that all predictor variables (WS, FB, RTK) are correlated, and assumes all directly influence the criterion variable (SD). Correlated variables are linked by undirected paths, which are labeled with the correlations. Table 2b presents the correlation matrix derived from our experiment. Multiple regression generates standard partial regression (beta) coefficients for each direct path between the predictor and criterion variables. These are  $-.291$ ,  $.81$  and  $-.2$  in Figure 4a. Each represents a standardized measure of the influence of one predictor variable on the criterion variable with the effects of the other predictor variables held constant. The resulting regression equation in standard format is  $\hat{SD} = .81 FB - .29 WS - .2 RTK$ . Because the regression coefficients are standardized they can be compared: a unit change in FB produces  $.81$  units change in SD, whereas a unit change in WS produces  $-.29$  units change in SD. FB is the stronger influence.

Figure 4a represents a decomposition of the correlations between SD and the other variables. The contributions can

<sup>9</sup> Pearl and Verma are developing efficient algorithms, related to path analysis, for causal induction (Pearl & Verma 1983).

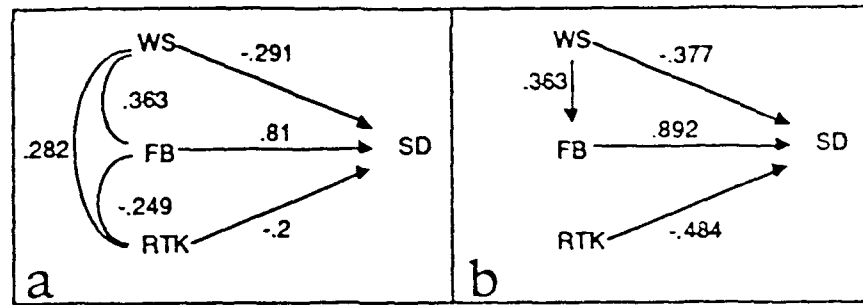


Figure 4: A Shows the Path Model Implicit in Multiple Regression. The Path Model in B Better Captures the Relationships Among These Variables in Phoenix.

be reconstituted by summing the influences along paths just as we did in Figure 3. Path analysis has three rules for identifying paths:

- 1) No more than one undirected link can be part of a path (e.g.,  $FB \rightarrow RTK \rightarrow SD$  is legal, but  $WS \rightarrow FB \rightarrow RTK \rightarrow SD$  isn't)
- 2) A path cannot go through a node twice.
- 3) A path can go backward on a directed link, but not after it has gone forward on another link (e.g.  $FB \leftarrow WS \rightarrow SD$  in Figure 4b is legal but  $\#PLANS \rightarrow FB \leftarrow WS$  in Figure 5 isn't).

The strength of each multilink path is just the product of its constituent coefficients, so the strength of the path  $FB \rightarrow RTK \rightarrow SD$  in Figure 4a is  $(-.249)(-.2) = .0498$ . The estimated correlation between a predictor and a criterion variable is the sum of the strengths of the paths that connect them. Thus

$$\begin{aligned} \hat{r}_{FBSD} &= .755 = .81 && \text{direct } FB \rightarrow SD \text{ path} \\ &+ (.363)(-.291) && FB \rightarrow WS \rightarrow SD \\ &+ (-.249)(-.2) && FB \rightarrow RTK \rightarrow SD \end{aligned}$$

So multiple regression follows the three steps of path analysis. First, propose a model, specifically, a model in which all predictor variables are correlated and directly linked to the criterion. Second, estimate path coefficients, specifically, calculate standard partial regression coefficients for the direct paths between predictor and criterion variables, and label the undirected links with the appropriate correlations. Third, estimate the correlations between each predictor and criterion variable by identifying legal paths between them, calculating the strength of each path, and summing the path strengths. In multiple regression, the estimated correlations are always identical to the actual correlations.

Multiple regression is a fine way to decompose correlations into their component influences *if you believe that multiple regression's implicit causal model represents your system*. Multiple regression is just path analysis on this implicit model, so if you don't believe the model you can propose another and run path analysis on it. This is what we did in Figure 4b. We know that WS and RTK are independent because our experiment varied them independently in a factorial design. (The reason they are correlated is the sampling bias identified as effect 2, above.) So we want to test a model in which WS influences SD directly and through FB, and RTK influences SD directly. The only question is how to estimate the path coefficients. The basic rules, which yield the coefficients in Figure 4b, are:

- 1) If W and X are uncorrelated causes of the criterion variable Y, then the path coefficients  $p_{YX}$  and  $p_{YW}$  are just the correlation coefficients  $r_{YX}$  and  $r_{YW}$ , respectively.
- 2) If W and X are correlated causes of the criterion variable Y, then the path coefficients  $p_{YX}$  and  $p_{YW}$  are the standard partial regression coefficients  $b'_{YX \cdot W}$  and  $b'_{YW \cdot X}$ , respectively, obtained from the regression of Y on X and W.

Is Figure 4b a better model than Figure 4a? We can answer the question in two ways. The statistical answer is that no model fits the data better, in terms of accounting for variance in the criterion variable, than the regression model. But this is hardly surprising when you consider that the regression model assumes everything influences everything else. The system analyst's answer is that we don't want models in which everything influences everything else; we want models in which some links are left out, in which causal influences are localized, not dissipated through a network of correlations. Let's ask, then, what it means for one such model to be

better than another. Again, the judgment depends on how well each accounts for the variance in the criterion variable and how accurately each estimates the correlations between variables, and, how well each represents what we surmise to be the causal structure of our system. Clearly, these criteria interact. We can imagine a model that fits the data well but cannot represent what we know to be the causal structure, but often we explore different plausible causal structures by seeing how well each fits the data.

The structure in Figure 5 represents one of our first guesses at the causal structure that relates WS, FPLAN and RTK to SD. We expected WS and FPLAN to each directly influence both #PLANS and FB, but neither to directly influence SD. We also expected RTK to influence #PLANS and SD directly. We thought #PLANS might influence FB and SD. We made these guesses based on regression analyses, the correlation matrix in Table 2b, some of the graphs shown earlier, and our general knowledge about how the Phoenix planner works.

After estimating the path coefficients as shown in Figure 5, we estimated the correlations  $\hat{r}_{SDi}$  between SD and each variable  $i$ . The estimates and the actual correlations are as follows:

	WS	FPLAN	RTK	#PLANS	FB
$\hat{r}_{SDi}$	.118	-.197	-.533	.719	.778
$r_{SDi}$	-.053	-.193	-.484	.713	.755

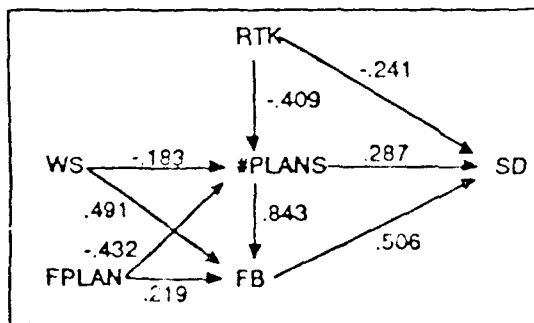


Figure 5: Path Model Relating Variables Influencing Shutdown Time.

Except for the disparity between the estimated and actual correlations between WS and SD, this model accounts pretty well for the actual correlations. At this point, we wanted to explain the influence of RTK on #PLANS. Why should decreasing RTK (slowing the Fireboss's thinking speed) increase the number of plans? One explanation is something like thrashing: There is always the possibility

that the environment will change in such a way that a plan is no longer appropriate, but this is much more likely when the environment changes rapidly relative to planning effort (i.e., when RTK is decreased). Thus, decreasing RTK means the Fireboss will have to throw away plans before they make much progress, resulting in an increase in #PLANS. To test this we introduced another variable, OVUT, which measures the percentage of time in a trial that the Fireboss spends planning. We expected OVUT to decrease with RTK, supporting the thrashing explanation. Figure 6 shows a modification of Figure 5, with the path  $RTK \rightarrow OVUT \rightarrow \#PLANS$  instead of  $RTK \rightarrow \#PLANS$ .

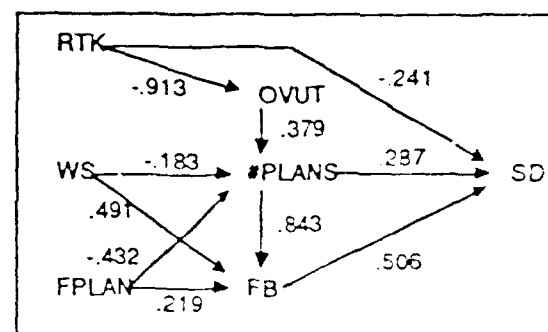


Figure 6: Adding the Endogenous Variable OVUT.

For this model, estimated correlations between SD and all the other variables are not appreciably different than they were for the model in Figure 5. But it appears that the variable OVUT does not add much to our understanding of thrashing, because it is completely determined by RTK. Consider what happens when we derive path coefficients for a slightly different model (Figure 7). In this case, OVUT has almost no influence ( $r_{OVUT \rightarrow \#PLANS} = .032$ ) on #PLANS. Recall, however, that this path coefficient is the standardized partial regression coefficient  $b'_{OVUT \rightarrow \#PLANS \cdot RTK}$ ; that is, the effect of OVUT on #PLANS with RTK held constant. The fact that this number is nearly zero means that OVUT has no effect on #PLANS when RTK is held constant; in other words, the effect of OVUT on #PLANS is due entirely to RTK.

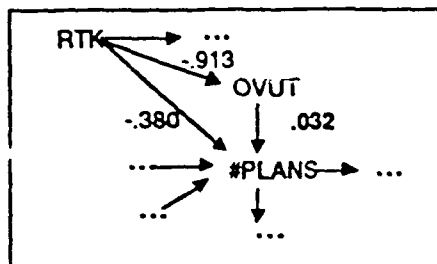


Figure 7: Showing the Effect of OVUT on #PLANS is Due Entirely to RTK.

## 6 CONCLUSION

We have presented results of an experiment with the Phoenix planner that confirm our predictions that its performance would be sensitive to some environmental conditions but not others. We have shown that the planner is not sensitive to variation in initial wind speed, a common environmental dynamic it faces. On the other hand, our results show that performance degrades as we change a fundamental relationship between the planner and its environment—the rate at which the Fireboss agent thinks. As we slowed the Fireboss's thinking speed in the experiment by decreasing RTK, performance degraded to the point where no plan succeeded on the first try. However, the planner was still able to succeed in many cases by replanning. While the success rate using replanning also degrades, replanning acts as a buffer, preventing the planner from failing catastrophically when it can't think fast enough to keep up with the environment. The data also show that replanning exerts a large influence on SD. We have presented a causal model, developed using path analysis, of the effects on SD of the various independent and endogenous variables we measured.

Replanning occurs when the environment doesn't match the Fireboss's expectations. In the current experiment, the rate at which the expectations became invalid was set by RTK. But the effect was indirect: Low RTK ensured that the Fireboss would be swamped (OVUT), which meant that bulldozers had to wait for instructions, which, in turn, increased the probability that they would not be able to carry out their instructions by their deadlines. This is what caused plans to fail. Environmental changes were only the instrument of the problem; RTK initiated it. But expectations, and thus plans, can also fail if the environment itself changes. We have yet to study whether replanning makes Phoenix robust against these changes, though our results with RTK suggest it does.

## Acknowledgements

This research is supported by DARPA under contract #F49620-89-C-00113; by AFOSR under the Intelligent Real-time Problem Solving Initiative, contract #AFOSR-91-0067; by NSF under an Issues in Real-Time Computing grant, #CDA-8922572; and by Texas Instruments Corporation. Thanks go to David Westbrook for invaluable design and programming help, Mike Sutherland for his statistical expertise, Scott Anderson for insightful comments on an early draft, and the many members of EKSL who have contributed to Phoenix. We also wish to thank an anonymous reviewer for a thorough and constructive reading. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

## References

- Asher, H.B. 1983. *Causal Modeling*. Sage Publications.
- Cohen, P.R., Hart, D.M. & St. Amant, R. 1992. Early warnings of plan failure, false positives, and envelopes: Experiments and a model. Dept. of Computer Science, Technical Report #92-20, University of Massachusetts, Amherst.
- Cohen, P.R. 1991. A survey of the Eighth National Conference on Artificial Intelligence: Pulling together or pulling apart? *AI Magazine* 12(1): 16-41.
- Cohen, P.R. 1990. Designing and analyzing strategies for Phoenix from models. *Proc. of the Workshop on Innovative Approaches to Planning, Scheduling and Control*. Pp. 9-21.
- Cohen, P.R., Greenberg, M.L., Hart, D.M. & Howe, A.E. 1989. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine* 10(3): 32-48.
- Hart, D.M., Anderson, S.D. & Cohen, P.R. 1990. Envelopes as a vehicle for improving the efficiency of plan execution. *Proc. of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*. Pp. 71-76.
- Howe, A.E. & Cohen P.R., 1990. Responding to environmental change. *Proc. of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*. Pp. 85-92.
- Li, C.C. 1975. *Path Analysis—A Primer*. Boxwood Press.
- National Wildlife Coordinating Group. Boise, Idaho. *NWCG Fireline Handbook*. November, 1985.
- Pearl, J. & Verma, T.S. 1991. A theory of inferred causation. *Principles of Knowledge Representation and Reasoning: Proc. of the Second International Conference*, J. Allen, R. Fikes, & E. Sandewall (Eds.). Morgan Kaufman. Pp. 441-452.

## **Appendix A: Textbook Prospectus**

### **Empirical Methods for Artificial Intelligence**

*Paul R. Cohen*

#### **I. Introduction**

- A. AI Programs as Objects of Empirical Studies
- B. Versions of the Three Basic Research Questions
- C. A Methodology for Achieving General Answers to the Basic Research Questions
- D. Kinds of Empirical Studies
- E. Data Analysis for the Four Classes of Studies
- F. Empirical AI in Context

#### **II. Exploratory Data Analysis**

##### **A. Data**

- A.1. Scales of Data
- A.2. Transforming data
- A.3. Measurement Theory
- A.4. Causal Models of Data Values

##### **B. Visualizing and Summarizing Data**

- B.1. Exploring One Variable
- B.2. Statistics for One Variable

##### **C. Joint Distributions**

- C.1. Joint Distributions of Categorical or Ordinal Variables.
  - C.1.1. Dependencies Among Row and Column Variables in Contingency Tables
- C.2. Contingency Tables for More than Two Variables
- C.3. Statistics for Joint Distributions of Categorical Variables
  - C.3.1. An Easy and Useful Special Case: 2 x 2 Table
- C.4. Visualizing Joint Distributions of Two Continuous Variables
  - C.4.1. Finding Hints of Causal Relationships in Scatterplots
  - C.4.2. Point Coloring to Find Potential Causal Factors
- C.5. Statistics for Joint Distributions of Two Continuous Variables
  - C.5.1. Pearson's Correlation Coefficient

##### **D. Time Series**

- D.1. Visualizing Time Series
  - D.1.1. Smoothing
- D.2. Statistics for Time Series

##### **E. Series of Categorical Variables. Behavior Traces**

- E.1. Visualizing Behavior Traces
- E.2. Statistics for Behavior Traces

#### **III. Fundamental Issues in Experiment Design**

- A. Some Terminology
- B. The Concept of Control
  - B.1. Control Conditions in MYCIN: A Case Study

- C. Four Spurious Effects
  - C.1. Ceiling and Floor effects
  - C.2. How to Detect Ceiling and Floor Effects
  - C.3. Regression Effects
  - C.4. Order Effects
- D. Sampling Bias
  - D.1. A Sampling Bias Due to an Arbitrary Cut-off
- E. The Dependent Variable
- F. Pilot Experiments
- G. Guidelines for Experiment Design
- H. Summary
- IV. Hypothesis Testing and Estimation.
  - A. Statistical Inference
  - B. Introduction to Hypothesis Testing
  - C. A More Formal View of Statistical Hypothesis Testing
    - C.1. Sampling Distributions
    - C.2. How to Get Sampling Distributions
      - C.2a. The Sampling Distribution of the Proportion
      - C.2b. The Sampling Distribution of the Mean
      - C.2c. The Standard Error of the Mean and Sample Size
      - C.2d. Standard Errors for Other Statistics
  - D. Parametric Tests of Hypotheses About Means
    - D.1. The Anatomy of the Z Test
    - D.2. Critical Values
    - D.3.  $p$  Values
    - D.4. When the Population Standard Deviation is Unknown
    - D.5. When  $N$  is Small: The  $t$  Test
      - D.5a. One Sample  $t$  Test
      - D.5b. Two Sample  $t$  Test.
      - D.5c. The Paired Sample  $t$  Test
  - E. Parameter Estimation and Confidence Intervals
    - E.1. Confidence intervals for  $\mu$  when  $\sigma$  is known.
    - E.2. Confidence intervals for  $\mu$  when  $\sigma$  is unknown.
    - E.3. An Application of Confidence Intervals: Error Bars
    - E.4. Hypothesis Testing with Confidence Intervals
  - F. Some Practical Issues
    - F.1. Reporting Confidence Intervals, When and Why.
    - F.2. Errors
    - F.3. How Big Should  $N$  Be?
  - G. Appendix 1
    - G.1. Degrees of Freedom
- V. Computer Intensive Hypothesis Testing
  - A. Monte carlo
  - B. Randomization
  - C. Bootstrap
  - D. Exact Nonparametric Tests

VI. Testing One-Factor Models

A. Case Studies of Experiments

- A.1. Addition Studies
- A.2. Ablation Studies
- A.3. Roughening Studies
- A.4. Minimal Pair Studies

B. Data Analysis for One Factor Experiments

- B.1. One-Way Analysis of Variance
- B.2. Simple Regression
- B.3. Tests on Learning Curves
- B.4. Dependency Detection

VII. Testing Multiple Factor Models.

A. Case Studies of Interactions between Architecture And Environment Factors on Behavior

B. Data Analysis for Multiple Factor Models

- B.1. Two- and Three-Way Analysis of Variance
- B.2. Multiple Regression
- B.3. Log-Linear Analysis

VIII. Building and testing causal models with path analysis.

IX. Tactics for Generalizing Results

- A. Finding Representative Problems
- B. Replicating Results